IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2014–11–19.
Submitted on 2014–07–18, accepted on 2014–08–05.
ISSN 2105–1232 © 2014 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2014.120

# Parameter-Free Fast Pixelwise Non-Local Means Denoising

Jacques Froment

Univ. Bretagne - Sud, UMR 6205, LMBA, F-56000 Vannes, France (Jacques.Froment@univ-ubs.fr)

*Communicated by* Bartomeu Coll *Demo edited by* Jacques Froment

**Abstract**

This article proposes a fast and open-source implementation of the well-known Non-Local Means (NLM) denoising algorithm, in its original pixelwise formulation. The fast implementation is based on the computation of patch distances using sums of lines that are invariant under a patch shift. The optimal parameters of NLM (in the average peak signal to noise ratio - PSNR - sense) are computed from an image database, thereby leading to a parameter-free NLM implementation. Comparison is performed with the parameter-free blockwise NLM implementation already proposed in IPOL journal by Buades, Coll and Morel. As expected the blockwise implementation offers better PSNR, at least when the noise standard deviation is large enough, but there is no significant difference in quality when performing visual inspection. The highlight is that the proposed parameter-free pixelwise NLM implementation is faster than the patchwise one by a factor of 6 to 49.

**Source Code**

The reviewed source code and documentation for the parameter-free fast pixelwise NLM algorithm are available from the web page of this article[1]. Compilation and usage instructions are included in the `README.txt` file of the archive.

**Keywords:** image denoising; non-local means

## 1 Introduction

The Non-Local Means (NLM) image denoising algorithm was introduced in 2005 by Antoni Buades, Bartomeu Coll and Jean-Michel Morel [1] and the success was such that this method has inspired a great number of variants and articles, see [3] for some updated references. Three factors largely explain why the image denoising community has been immersed in the NLM approach: the original algorithm remains simple; it provides great visual quality; it introduces a basic tool to exploit the non-local redundancy of natural images. Despite the subsequent introduction of more efficient denoising algorithms (see [20, 16] for recent comparisons between NLM and some state of the art

[1] https://doi.org/10.5201/ipol.2014.120

denoising schemes), NLM remains a reference method for image denoising. It is therefore essential that the image denoising community has freely available a documented, verified and open-source implementation of NLM in its original version.

Despite the number of NLM codes that can be found in the Internet, it seems that there are only two implementations that fulfill these conditions: the `nlmeans` module included in MegaWave2 [11] and written by Lionel Moisan and the IPOL article [4] from Buades, Coll and Morel. The first code implements the pixelwise NLM algorithm given in [1] (Section 5.1) for gray-level images only and using the Euclidean norm instead of the uniform one to define neighbor patches. The parameters' default values correspond to those in [1], but they turn out to be non optimal for most images. Finally, this code does not implement any algorithmic trick or parallelization of computing, making it a slow running program. While the IPOL article [4] describes both the pixelwise NLM (NLM-P) ([1] Section 5.1) and the blockwise NLM (NLM-B) ([1] Section 5.5.2), the published demo and code implement the blockwise version only[2].

The present article aims to propose a fast and parameter-free implementation of the pixelwise NLM as described in [1] (pages 510-512). In order to fix the notation, let us first recall the equations that lead to this algorithm.

The input, noisy image $v$, is assumed to come from the classical additive noise model

$$v(x) = u(x) + \epsilon(x),\ x \in \Omega, \tag{1}$$

where $u$ is the original image, $\Omega$ is the set of pixels and $\epsilon$ is the noise perturbation: $(\epsilon(x))_{x\in\Omega}$ are independent and identically distributed (i.i.d.) Gaussian variables with mean 0 and standard deviation $\sigma > 0$. In the case of gray-level images, $u$ and $v$ take real values (integer values in the discrete model) whereas for color images, $u$ and $v$ are vector-valued.

The output, denoised image, is the estimator $\tilde{u}$ computed as

$$\tilde{u}(x) = \sum_{y\in\Omega_x} w(x,y)v(y), \tag{2}$$

where the weight $w(x,y)$ estimates the similarity between the pixel $x$ and $y$ in the original image and the sequence of weights satisfies

$$w(x,y) \geq 0 \ \text{ and } \ \sum_{y\in\Omega_x} w(x,y) = 1,\ \forall x \in \Omega, y \in \Omega_x. \tag{3}$$

The set of pixels $\Omega_x$ is a neighborhood of $x \in \Omega$, it is defined as

$$\Omega_x = \{y \in \Omega : \|x - y\|_\infty \leq D\}, \tag{4}$$

for $D > 0$ a fixed size parameter. The window $\Omega_x$ is called the search window at $x$ and, for simplicity and faster computations, in [1] a choice of a square shape of fixed size is made while, according to authors, the search window should cover the entire image plane, hence the non-local nature of the algorithm. However, it has been reported that using for NLM a neighborhood instead of the whole image plane allows to increase the denoising performance [12, 13, 21, 22], see also the discussion in [9] and the specific study in [19] where it is experimentally established that the optimal window size $D$ is very small, when using a variant of the pixelwise NLM. As this present article will establish the best parameters, it will give an answer for the original pixelwise NLM.

This is the following particular form of weights $w(x,y)$ that makes the difference between NLM and classical denoising methods known as neighborhood filters:

$$w_{\text{NLM-Pa}}(x,y) = \frac{1}{n(x)} e^{-\frac{\|V(x)-V(y)\|^2_{2,a}}{h^2}}, \tag{5}$$

[2]Blockwise NLM is called patchwise NLM in [4].

where $n(x)$ is a generic normalization factor and $h > 0$ is a filtering parameter. We write here the denominator of the fraction in the exponential as $h^2$, thus following the notation in [3, 1, 4], whereas one reads $2h^2$ in most subsequent articles about NLM.

The estimation of the oracle $|u(x) - u(y)|$ is performed using the Gaussian Euclidean norm $\|V(x) - V(y)\|_{2,a}$, where $a$ is the standard deviation of the Gaussian and where $V(x)$ is the patch of size $d \times d$ centered at $x$ in image $v$:

$$V(x) = \{v(y) : y \in \Omega, \|x - y\|_\infty \leq d\}. \tag{6}$$

Another significant difference with some subsequent articles is in the fact that weights are computed without subtracting $2\sigma^2$ to the square of this norm. Such an alternative is written

$$w_{\text{NLM-Pa-variant}}(x, y) = \frac{1}{n(x)} e^{-\frac{\max\{\|V(x)-V(y)\|_{2,a}^2 - 2\sigma^2, 0\}}{h^2}} \tag{7}$$

and this variant is motivated by the fact that

$$E\{\|V(x) - V(y)\|_{2,a}^2\} = \|U(x) - U(y)\|_{2,a}^2 + 2\sigma^2, \tag{8}$$

where the expectation is taken over the noise distribution. The choice to consider (5) rather than (7) is related to the concern to implement the original NLM algorithm. Note that in [4] and unlike [1, 3], the authors *do subtract* $2\sigma^2$ to the square of the norm.

In the discrete setting, the Gaussian Euclidean norm writes

$$\|V(x) - V(y)\|_{2,a}^2 = \sum_{\{z \in \mathbb{Z}^2 : \|z\|_\infty \leq d_s\}} K_{G_a}(z) \, \|v(x+z) - v(y+z)\|_2^2 \tag{9}$$

with

$$K_{G_a}(z) = \frac{e^{-\frac{\|z\|_2^2}{2a^2}}}{\displaystyle\sum_{\{t \in \mathbb{Z}^2 : \|t\|_\infty \leq d_s\}} e^{-\frac{\|t\|_2^2}{2a^2}}} \tag{10}$$

and $d_s = (d-1)/2$ being the patch side half-length, $d$ being an odd number. In the right-hand of Equation (9) the symbol $\|v(x)\|_2$ must be understood as the Euclidean norm of $v(x)$, which is simply the absolute value of $v(x)$ in the case of a gray-level image. In the case of a color image, $v(x)$ is a vector of $N_c$ components, $N_c$ being the number of channels, and $\|v(x)\|_2^2$ is the sum of the squares of each component.

We denote by NLM-Pa this pixelwise NLM denoising scheme, given by equations (2) and (5). The letter "a" recalls the use of the Gaussian Euclidean norm (9) of parameter $a$. Besides the original articles on NLM [3, 1], the Gaussian kernel is often dropped and the norm between patches becomes the standard Euclidean one, being in the discrete case the mean square error between $V(x)$ and $V(y)$:

$$\|V(x) - V(y)\|_2^2 = \frac{1}{d^2} \sum_{\{z \in \mathbb{Z}^2 : \|z\|_\infty \leq d_s\}} \|v(x+z) - v(y+z)\|_2^2 \,. \tag{11}$$

In such a case, NLM-weights write

$$w_{\text{NLM-P}}(x, y) = \frac{1}{n(x)} e^{-\frac{\|V(x)-V(y)\|_2^2}{h^2}} \,. \tag{12}$$

We denote by NLM-P the pixelwise NLM denoising scheme with the Euclidean norm instead of the Gaussian Euclidean one. When mentioned in the literature, the reason for using NLM-P rather than

NLM-Pa is that the increase in quality provided by the Gaussian kernel is low, while it requires the additional parameter $a$ to be tuned. However, it seems that there is no experimental study using image databases to quantify the effect of the Gaussian kernel. It is a goal of this article to examine to what extend the previous assertion is valid.

To unify the presentation of NLM-P and NLM-Pa, we may denote by $K$ the kernel used to weight the norm between patches ($K = K_{G_a}$ for NLM-Pa, $K = 1/d^2$ for NLM-P) and the corresponding norm will be written $\|.\|_{2,K}$. Before considering optimization, let us present the basic algorithm.

## 2 Basic Algorithm for the Pixelwise NLM

The pixelwise NLM method may be implemented in a straightforward manner using the above equations. To denoise a pixel $x \in \Omega$, one has to compute the patch $V(y)$ for all $y \in \Omega_x$ and therefore to scan each pixel $z$ such that $\|y - z\|_\infty \leq d$. A window of size $(D + d)^2$ centered in $x$ is then to be accessed and this requires adding to the borders of the input image $v$ a strip of width $\frac{D-1}{2} + \frac{d-1}{2} = D_s + d_s$. As usual in image processing, the padding is performed by mirror reflection and the resulting symmetrized image will be denoted `Vsym` in the following pseudo-codes (to assist the reader, Table 1 lists the main notations specifying those used in the mathematical context and those used for pseudo and source codes). The kernel $K$ is precomputed as an array of size $d^2$ using (10) for NLM-Pa (or as a constant $K = 1/d^2$ for NLM-P).

The main loop consists in going through each pixel $x \in \Omega$ and as explained in the next section, parallelization may be performed at this stage by assigning a specific $x$ to a specific thread. A first pass of all neighboring pixels $y \in \Omega_x$ is performed in order to compute the array of NLM-weights, using (5) and (9) or (12) and (11). A second pass of all neighboring pixels $y \in \Omega_x$ is performed to get the denoised pixel $\tilde{u}(x)$ using the estimation in (2). Note that, in case of gray-level images, these two passes can be merged together. A more detailed sketch of this algorithm is proposed as pseudo-code in Algorithm 1.

From this pseudo-code, one can easily deduce the complexity of the pixelwise NLM in its basic implementation. Let us count the number of operations following the standard uniform cost model [27]. Let $N = |\Omega| = N_1 N_2$ be the number of pixels of the input image $v$, $N_c$ the number of color channels, $N_S = (N_1 + 2(D_s + d_s))(N_2 + 2(D_s + d_s))$ the number of pixels of the symmetrized image and $c$ a generic constant. The symmetrization of the input image requires $cN_sN_c$ operations and the precomputation of the kernel $K$, $cd^2$ operations. Inside the main loop $x \in \Omega$, the first pass of all neighboring pixels $y \in \Omega_x$ used to compute NLM-weights requires $cN_cD^2d^2$ operations, while the second pass that computes the estimator $\tilde{u}(x)$ needs $cN_cD^2$ operations only. Therefore and using the big O notation to hide constant factors and smaller terms, the complexity of the plain pixelwise NLM is given by that of the first pass of the main loop, that is $O(NN_cD^2d^2)$.

## 3 Fast Algorithms for the Pixelwise NLM

Different strategies can be deployed to increase the speed of pixelwise NLM. First, it should be noted that the algorithm is particularly well suited to parallel computing, thanks to the independent processing of each pixel $x \in \Omega$ to be denoised. The easiest way to implement parallel computations is therefore to assign, in the main loop level, a specific pixel $x$ to a specific thread. This allows to roughly divide the execution time by the number of available threads (the gain is actually a bit lower due to input/output processes and because of the initialization pass, although this one may also be parallelized). In the code associated to this article, parallelization is implemented in the main loop level only, both for the plain and the proposed fast implementations.

| Mathematical notation | Code notation | Comments |
|---|---|---|
| $d$ | `d` | Patch side length is `d=2*ds+1` |
| $d_s$ | `ds` | Patch side half-length |
| $d^2$ | `d2` | Number of pixels of the patch |
| $D$ | `D` | Search window side length is `D=2*Ds+1` |
| $D_s$ | `Ds` | Search window side half-length |
| $D^2$ | `D2` | Number of pixels of the search window $D^2 = \|\Omega_x\|$ |
| $\|V(x) - V(y)\|_2^2$ or $\|V(x) - V(y)\|_{2,a}^2$ | `Dist2` | Euclidean or Gaussian Euclidean norm between two patches, see (11) and (9) |
| $K$ | `K` | 2D-Kernel of weighted Euclidean norm, see (10) |
| $K_1, K_2$ | `K` | 1D-Kernel of weighted Euclidean norm (fast NLM-Pa using sum of invariant lines), see (20) |
| $L_K^2$ | `L2` | Weighted Euclidean distance between two patches at one line (NLM-Pa using sum of invariant lines), see (22) |
| $N$ | `N` | Number of pixels of the input image $N = \|\Omega\| = N_1 \times N_2$ |
| $N_1$ | `N1` | Number of columns of the input image ($x_1$ axis) |
| $N_2$ | `N2` | Number of rows of the input image ($x_2$ axis) |
| $N_c$ | `Nc` | Number of channels (planes) in the input image (1 or 3) |
| | `NNc` | Image's size N × Nc |
| $N_S$ | `NS` | Number of pixels of the symmetrized image: $N_S = (N_1 + D_s + d_s) \times (N_2 + D_s + d_s)$ |
| $\sigma$ | `sigma` | Noise standard deviation |
| $u$ | `U` | Input noiseless image |
| $\tilde{u}$ | `Vd` | Output denoised image |
| - | `Vd[c]` | Color channel #c of the denoised image (pseudo-code) |
| $v$ | `V` | Input noisy image |
| - | `Vsym` | Symmetrized noisy image |
| - | `Vsym[c]` | Color channel #c of `Vsym` (pseudo-code) |
| $w$ | `W` | NLM-weights image |

Table 1: Main mathematical notations (used in this PDF article) and main code notations (for pseudo-codes in this PDF article and/or source codes) and correspondence between them.

Using parallelization should be regarded as a technical trick only, insofar as it does not reduce the algorithmic complexity. More interesting strategies seek to change the way the calculations are carried out, so that the complexity may be effectively reduced. As seen before, the biggest term that sets the complexity of the plain pixelwise NLM is given by the first pass of the main loop that computes the distance between patches $V(x)$ and $V(y)$. Therefore, strategies are committed to reduce the cost associated with this distance computation.

The most known approaches involve estimating patches $V(y)$ for which distance to $V(x)$ is probably significant or, more generally, which are dissimilar to $V(x)$ in some sense. In such a case the pixel $y$ is simply removed in the neighborhood $\Omega_x$ or, equivalently, the weight $w(x, y)$ is set to 0 and the exact distance (9) or (11) is not computed. To be useful from the point of view of complexity, the preselection of similar patches has to be carried out with less than $cN_cD^2d^2$ operations. Such approaches have been proposed in [17] (preselection using mean and gradient similarities), [5] (using first and second moments), [18] (using conditional probabilities and critical pixels), [23] (using a multi-resolution decomposition). It is important to emphasize that these approaches do not implement the exact pixelwise NLM: the denoised image is different to the one obtained using Algorithm 1. For this reason and even if the denoised image could be of better quality, such patches preselection

**Algorithm 1:** Pseudo-code for pixelwise Non-Local Means (NLM-P and NLM-Pa): basic algorithm

**input** : `V`, `ds`, `Ds`, `h`, `a`
**output**: `Vd`
**** INITIALIZATION ****
(`N1`, `N2`) ← image size
`Nc` ← number of color channels
`Vsym` ← symmetrized noisy image V with border Ds+ds
**if** $a > 0$ **then** *Kernel for NLM-Pa*
  `K` ← 2D-kernel of Gaussian Euclidean norm, computed using (10)
**else** *Kernel for NLM-P*
  `K` ← constant kernel $1/d^2$ for $d = 2 \times \texttt{ds} + 1$
**** MAIN LOOP *** denoise pixel $x = (x_1, x_2)$, center of the 1st patch*
**for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
  *— FIRST PASS — compute NLM-weights*
  **for** $y = (y1, y2) = (x1 - Ds, x2 - Ds)$ *to* $(x1 + Ds, x2 + Ds)$ **do**
    *$y = (y_1, y_2)$ is the center of the 2d patch*
    *Compute distance between the two patches following (11) (NLM-P) or (9) (NLM-Pa)*
    Dist2 ← 0
    **for** $c = 0$ *to* $Nc - 1$ **do**
      **for** $z = (z1, z2) = (-ds, -ds)$ *to* $(+ds, +ds)$ **do**
        Dist2 ← Dist2 + $K(z) \times (\texttt{Vsym}[c](x + z) - \texttt{Vsym}[c](y + z))^2$
    *Compute unnormalized weight $w(x, y)$ following (5) and (12)*
    $W(x, y) = e^{-\text{Dist2}/(Nc \times h^2)}$
  *— SECOND PASS — compute denoised pixel $\tilde{u}(x)$*
  **for** $c = 0$ *to* $Nc - 1$ **do**
    $r \leftarrow 0$ *Sum of weighted pixel's values, without normalization*
    $s \leftarrow 0$ *Sum of weights, for normalization*
    **for** $y = (y1, y2) = (x1 - Ds, x2 - Ds)$ *to* $(x1 + Ds, x2 + Ds)$ **do**
      $r \leftarrow r + W(x, y) \times \texttt{Vsym}[c](y)$ *The weighted average is done here*
      $s \leftarrow s + W(x, y)$ *Compute sum of weights, for normalization*
    *Final estimate based on the assumption that original planes take values in $[0, 255]$*
    `Vd`$[c](x) \leftarrow \min(\max(r/s, 0), 255)$

is not considered in this article that aims to implement and study the original pixelwise NLM.

## 3.1 Fast NLM-P Using Integral Images

A strategy for reducing the complexity of patch distance computations while maintaining exact calculations is described in [25] and [7]. The method is known as *integral images* [24] (or *summed area tables* [6] in the context of texture mapping) and it allows to efficiently compute the sum of values of any image in a rectangular subset of a grid. A recent review of the integral image algorithm and its application is proposed in [10].

In the context of patch distance, the image to be summed takes the form

$$s_t(z) = \|v(z) - v(z + t)\|_2^2, \quad z = (z_1, z_2) \in \Omega \tag{13}$$

where $t = y - x \in [\![-D_s, +D_s]\!]^2$ is a translation vector. The associated integral image (or summed area table) is given by

$$S_t(x) = \sum_{\{z=(z_1,z_2)\in\mathbb{N}^2 : 0\le z_1\le x_1, 0\le z_2\le x_2\}} s_t(z), \quad x = (x_1, x_2) \in \Omega, \tag{14}$$

with symmetric or periodic extension at the image boundaries. Note that the integral image can be calculated in $O(NN_c)$ operations using the recursive sequence

$$\forall x = (x_1, x_2) \in \Omega, x_1 \ge 1, x_2 \ge 1, \; S_t(x) = s_t(x) + S_t(x_1-1, x_2) + S_t(x_1, x_2-1) - S_t(x_1-1, x_2-1). \tag{15}$$

The key point is that the Euclidean norm (11) may be written

$$\begin{aligned} \|V(x) - V(y)\|_2^2 = \frac{1}{d^2} \big( \quad & S_t(x_1 + d_s, x_2 + d_s) + S_t(x_1 - d_s, x_2 - d_s) \\ & - S_t(x_1 + d_s, x_2 - d_s) - S_t(x_1 - d_s, x_2 + d_s) \quad \big) . \end{aligned} \tag{16}$$

Hence the fast NLM-P algorithm using integral images may work as follows: first, all integral images are computed using (15). As there are $D^2 = (2D_s+1)^2$ translations $t = y-x$, this initialization pass needs $O(NN_cD^2)$ operations. In the first pass of the main loop, the distance between two patches is computed in constant time using (16). The remaining of the algorithm is identical to the basic version, see Algorithm 2 for the adapted pseudo-code. A noticeable fact of this fast algorithm is that the computation of the NLM-weights is now independent to the patch size. The total cost of the fast NLM-P algorithm using integral images is therefore determined by that of the initialization pass and the second pass of the main loop, that is $O(NN_cD^2)$. Note that this implementation, the closest to that of the basic one, uses lot of memory since it requires recording all integral images $S_t$ for $t \in [\![-D_s, +D_s]\!]^2$ that is, $D^2$ times the size of the input image.

To reduce the memory size, a solution is to change the order of loops so that the main loop becomes the shift vector $t \in [\![-D_s, +D_s]\!]^2$. One then no longer needs to index the integral image $S_t$ by the parameter $t$. However, it becomes necessary to allocate an array to compute the sum of weights needed for the normalization. Finally, the additional memory size required by the integral images implementation is only two times the size of the input image, see Algorithm 3 for the corresponding pseudo-code. Note that, due to the partial calculation of the estimate in the main loop, this version optimized for memory usage would be less suitable for parallel processing (it would require threads to be synchronized to protect access to shared data).

A drawback of this integral images approach is that it does not directly allow the computation of a weighted norm using a kernel $K$, as the Gaussian one in (9). Another problem may occur even in the simple case of NLM-P: when image size as well as patch distances are large, some values of the integral image (14) may become so huge that the accuracy of the final result may drop, even when using a double precision representation (see [10] for a discussion of this phenomenon). Therefore this fast algorithm can hardly be used to implement NLM-Pa, unlike the two ones that will be now presented.

## 3.2 Fast NLM-Pa Using FFT

In [8], C-A. Deledalle, V. Duval and J. Salmon introduce a fast algorithm to compute the pixelwise NLM using the Fast Fourier Transform (FFT). Their algorithm has a broader scope since it considers patches of arbitrary shapes, but it can obviously be applied in the particular case of square patches. The computational complexity decrease is achieved by two modifications. First, loops are rearranged so that one considers all pixels $x$ for all translation vectors $t \in [\![-D_s, +D_s]\!]^2$, in a manner similar

**Algorithm 2:** Pseudo-code for a fast NLM-P: integral images algorithm, version 1 (without memory optimization)

**input** : `V`, `ds`, `Ds`, `h`
**output**: `Vd`
**Function** *st(t; z) Compute $s_t(z)$ following (13)*
  Dist2 $\leftarrow 0$
  **for** $c = 0$ *to* $Nc - 1$ **do**
    Dist2 $\leftarrow$ Dist2 + $(\texttt{Vsym}[c](z) - \texttt{Vsym}[c](z+t))^2$
  **return** *(Dist2)*
**** INITIALIZATION ****
(`N1`, `N2`) $\leftarrow$ image size
`Nc` $\leftarrow$ number of color channels
`Vsym` $\leftarrow$ symmetrized noisy image V with border Ds+ds
*Compute integral images S[t] for all t, following (15)*
**for** $t = (t1, t2) = (-Ds, -Ds)$ *to* $(+Ds, +Ds)$ **do**
  S[t](0,0) $\leftarrow 0$
  **for** $x1 = 1$ *to* $N1 - 1$ **do**
    S[t](x1,0)=st(t;(x1,0))+S[t](x1-1,0)
  **for** $x2 = 1$ *to* $N2 - 1$ **do**
    S[t](0,x2)=st(t;(0,x2))+S[t](0,x2-1)
  **for** $x = (x1, x2) = (1, 1)$ *to* $(N1 - 1, N2 - 1)$ **do**
    S[t](x)=st(t;x)+S[t](x1-1,x2)+S[t](x1,x2-1)-S[t](x1-1,x2-1)
**** MAIN LOOP *** denoise pixel $x = (x_1, x_2)$, center of the 1st patch*
**for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
  *— FIRST PASS — compute NLM-weights*
  **for** $y = (y1, y2) = (x1 - Ds, x2 - Ds)$ *to* $(x1 + Ds, x2 + Ds)$ **do**
    *$y = (y_1, y_2)$ is the center of the 2d patch*
    *Compute distance between the two patches using integral images, see (16)*
    Dist2 $\leftarrow S[t](x + (ds, ds)) + S[t](x - (ds, ds)) - S[t](x + (ds, -ds)) - S[t](x + (-ds, +ds))$
    Dist2 $\leftarrow$ Dist2 $/d^2$
    *Compute unnormalized weight $w(x, y)$ following (12)*
    $W(x, y) = e^{-\text{Dist2}/(Nc \times h^2)}$
  *— SECOND PASS — compute denoised pixel $\tilde{u}(x)$*
  **for** $c = 0$ *to* $Nc - 1$ **do**
    $r \leftarrow 0$ *Sum of weighted pixel's values, without normalization*
    $s \leftarrow 0$ *Sum of weights, for normalization*
    **for** $y = (y1, y2) = (x1 - Ds, x2 - Ds)$ *to* $(x1 + Ds, x2 + Ds)$ **do**
      $r \leftarrow r + W(x, y) \times \texttt{Vsym}[c](y)$ *The weighted average is done here*
      $s \leftarrow s + W(x, y)$ *Compute sum of weights, for normalization*
    *Final estimate based on the assumption that original planes take values in $[0, 255]$*
    $\texttt{Vd}[c](x) \leftarrow \min(\max(r/s, 0), 255)$

to the trick used in Algorithm 3 to avoid recording all integral images. Second, given such a $t$ the

**Algorithm 3:** Pseudo-code for a fast NLM-P: integral images algorithm, version 2 (with memory optimization)

**input** : `V`, `ds`, `Ds`, `h`
**output**: `Vd`
**Function** *st(t; z) Compute $s_t(z)$ following (13)*
  Dist2 ← 0
  **for** $c = 0$ *to* $Nc - 1$ **do**
    Dist2 ← Dist2 + $(\texttt{Vsym}[c](z) - \texttt{Vsym}[c](z+t))^2$
  **return** *(Dist2)*
**** INITIALIZATION ****
(`N1`, `N2`) ← image size
`Nc` ← number of color channels
`Vsym` ← symmetrized noisy image V with border Ds+ds
`Vd` ← all planes filled with 0
$SW$ ← array filled with 0 *Sum of Weights image*
**** MAIN LOOP *** shift vector* $t = (t1, t2)$
**for** $t = (t1, t2) = (-Ds, -Ds)$ *to* $(+Ds, +Ds)$ **do**
  *— Step 1 — Compute the integral image St, t being fixed, following (15)*
  St(0,0) ← 0
  **for** $x1 = 1$ *to* $N1 - 1$ **do**
    St(x1,0)=st(t;(x1,0))+St(x1-1,0)
  **for** $x2 = 1$ *to* $N2 - 1$ **do**
    St(0,x2)=st(t;(0,x2))+St(0,x2-1)
  **for** $x = (x1, x2) = (1, 1)$ *to* $(N1 - 1, N2 - 1)$ **do**
    St(x)=st(t;x)+St(x1-1,x2)+St(x1,x2-1)-St(x1-1,x2-1)
  *— Step 2 — Compute weight and estimate for patches* $V(x), V(y)$ *with* $y = x + t$
  **for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
    $y \leftarrow x + t$
    *Compute distance between the two patches using integral images, see (16)*
    Dist2 ← $St(x + (ds, ds)) + St(x - (ds, ds)) - St(x + (ds, -ds)) - St(x + (-ds, +ds))$
    Dist2 ← Dist2 $/d^2$
    *Compute unnormalized weight* $w(x, y)$ *following (12)*
    $W(x, y) = e^{-\text{Dist2}/(Nc \times h^2)}$
    $SW(x) \leftarrow SW(x) + W(x, y)$ *Compute sum of weights, for subsequent normalization*
    *Compute estimate as weighted average*
    **for** $c = 0$ *to* $Nc - 1$ **do**
      $\texttt{Vd}[c](x) \leftarrow \texttt{Vd}[c](x) + W(x, y) \times \texttt{Vsym}[c](y)$
**** FINAL LOOP *** Compute final estimate at pixel* $x = (x_1, x_2)$
**for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
  **for** $c = 0$ *to* $Nc - 1$ **do**
    $\texttt{Vd}[c](x) \leftarrow \min(\max(\texttt{Vd}[c](x)/SW(x), 0), 255)$

weighted norm of patches differences is written as a discrete convolution product

$$\|V(x) - V(x+t)\|^2_{2,K} = \sum_{\{z \in \mathbb{Z}^2 : \|z\|_\infty \le d_s\}} K(z) \, \|v(x+z) - v(x+t+z)\|_2^2 = (\tilde{K} * s_t)(x) \qquad (17)$$

where $\tilde{K}(z) = K(-z)$, $*$ is the discrete convolution product and $s_t$ is the square difference image defined in (13). As it is well known, the convolution product may be computed in $O(NN_c \log(NN_c))$ operations using the 2D discrete Fourier transform (2D-FFT) denoted by $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$:

$$\|V(x) - V(x+t)\|^2_{2,K} = \mathcal{F}^{-1}(\mathcal{F}(\tilde{K})\mathcal{F}(s_t))(x). \tag{18}$$

Apart from this patch distance computation done using 2D-FFT rather than the integral images concept, the structure of this method, detailed in Algorithm 4, is similar to Algorithm 3. The final complexity of NLM-Pa using FFT, given by the one of its main loop where (18) has to be computed for any translation vectors $t \in [\![-D_s, +D_s]\!]^2$, is $O(NN_cD^2 \log(NN_c))$. Thus, as with integral images, the computation of the NLM-weights is independent from the patch size. However, the term $O(\log(NN_c))$ emerges as a cost to pay to weight the Euclidean norm using a kernel $K$ and this can be a major drawback for denoising images of large sizes.

### 3.3 Fast NLM-Pa Using Sums of Invariant Lines (SIL)

In what follows is proposed the SIL algorithm for NLM-Pa whose complexity is $O(NN_cD^2d)$ instead of $O(NN_cD^2 \log(NN_c))$, which is therefore faster than the FFT approach for sufficiently large images. The only condition is for the kernel $K$ to be a separable function, that is

$$\forall z = (z_1, z_2) \in \mathbb{Z}^2, \|z\|_\infty \leq d_s,\ K(z) = K_1(z_1)K_2(z_2). \tag{19}$$

This condition is satisfied with the Gaussian kernel used in NLM-Pa, for which we have

$$\forall i \in \mathbb{Z}, |i| \leq d_s,\ K_1(i) = K_2(i) = \frac{e^{-\frac{i^2}{2a^2}}}{\sum\limits_{\{t\in\mathbb{Z}:|t|\leq d_s\}} e^{-\frac{t^2}{2a^2}}}. \tag{20}$$

In such case, calculating the patch distance can use the following trick: a shift of one row (or one column) of the two patches does not require recalculation of all pixel differences; previous differences may be kept and new pixel differences have to be computed for the new incoming row (or column) only. In what follows the shift on rows has been implemented, but a similar result would be obtained with a shift on columns.

Using this trick goes through the decomposition of the weighted quadratic distance such as (9) to a sum of lines that are invariant under a patch shift: for $t \in [\![-D_s, +D_s]\!]^2$ a translation vector,

$$\|V(x) - V(x+t)\|^2_{2,K} = \sum_{\{z_2\in\mathbb{Z}:|z_2|\leq d_s\}} K_2(z_2)L^2_K(x,t,z_2) \tag{21}$$

where $L^2_K(x,t,z_2)$ is the weighted quadratic distance between patches $V(x)$ and $V(x+t)$ at the line index $z_2$:

$$L^2_K(x,t,z_2) = \sum_{\{z_1\in\mathbb{Z}:|z_1|\leq d_s\}} K_1(z_1)\left\|v(x+z) - v(x+t+z)\right\|^2_2. \tag{22}$$

The shift invariance property comes from the fact that, for $l = (0,1)$ a translation vector of one line down (as usual in image processing, the $X_2$-axis is assumed to point down),

$$\forall z_2 \in \mathbb{Z}, -d_s \leq z_2 < +d_s,\ L^2_K(x,t,z_2) = L^2_K(x-l,t,z_2+1). \tag{23}$$

Therefore, (21) may be written

$$\|V(x) - V(x+t)\|^2_{2,K} = K_2(d_s)L^2_K(x,t,d_s) + \sum_{z_2=-d_s}^{d_s-1} K_2(z_2)L^2_K(x-l,t,z_2+1). \tag{24}$$

**Algorithm 4:** Pseudo-code for a fast NLM-Pa: FFT method

**input** : V, ds, Ds, h, a
**output**: Vd
**** INITIALIZATION ****
(N1, N2) ← image size
Nc ← number of color channels
Vsym ← symmetrized noisy image V with border Ds+ds
Vd ← all planes filled with 0
$SW$ ← array filled with 0 — *Sum of Weights image*
K ← kernel of Gaussian Euclidean norm, computed using (10)
FK ← FFT2D(K) — *Precomputation of $\mathcal{F}(K)$, note that $K = \tilde{K}$ since $K = K_{G_a}$*
**** MAIN LOOP *** shift vector $t = (t1, t2)$*
**for** $t = (t1, t2) = (-Ds, -Ds)$ *to* $(+Ds, +Ds)$ **do**
  — *Step 1 — Compute the Patch Distance image for a given t, PDt, following (18)*
  **for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
    *Compute the Square difference image st, t being fixed, following (13)*
    $\mathrm{st}(x) \leftarrow 0$
    **for** $c = 0$ *to* $Nc - 1$ **do**
      $\mathrm{st}(x) \leftarrow \mathrm{st}(x) + (\mathtt{Vsym}[c](x) - \mathtt{Vsym}[c](x + t))^2$
  Fst ← FFT2D(st) — *Computation of $\mathcal{F}(st)$*
  **for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
    FPDt($x$)= FK($x$) × Fst($x$) — *Get the Fourier transform of the Patch Distance image*
  PDt ← IFFT2D(FPDt) — *Inverse 2D-FFT to end (18)*
  — *Step 2 — Compute weight and estimate for patches $V(x), V(y)$ with $y = x + t$*
  **for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
    $y \leftarrow x + t$
    $W(x, y) = e^{-\mathrm{PDt}(x)/(Nc \times h^2)}$ — *Compute unnormalized weight $w(x, y)$ following (5)*
    $SW(x) \leftarrow SW(x) + W(x, y)$ — *Compute sum of weights, for subsequent normalization*
    *Compute estimate as weighted average*
    **for** $c = 0$ *to* $Nc - 1$ **do**
      $\mathtt{Vd}[c](x) \leftarrow \mathtt{Vd}[c](x) + W(x, y) \times \mathtt{Vsym}[c](y)$
**** FINAL LOOP *** Compute final estimate at pixel $x = (x_1, x_2)$*
**for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
  **for** $c = 0$ *to* $Nc - 1$ **do**
    $\mathtt{Vd}[c](x) \leftarrow \min(\max(\mathtt{Vd}[c](x)/SW(x), 0), 255)$

Note that, if $K_2$ would be a constant (such as with NLM-P), one would have

$$\|V(x) - V(x + t)\|^2_{2,K} = \|V(x - l) - V(x + t - l)\|^2_{2,K} + K_2 L^2_K(x, t, d_s) - K_2 L^2_K(x - l, t, -d_s). \quad (25)$$

Nevertheless, the property (24) is sufficient to gain optimization as long as one records the array $L^2_K$: if we assume values of $\{L^2_K(x - l, t, z)\}_{z=-d_s+1,\ldots,d_s}$ saved while computing the distance between patches $V(x - l)$ and $V(x + t - l)$, the computation of the distance between the one row shifted patches $V(x)$ and $V(x + t)$ requires the evaluation of the distance at the new incoming row index $d_s$ only, given by $L^2_K(x, t, d_s)$. In this way, for a fixed $x$ and $y$ the computation of (24) needs $O(N_c d)$ operations only and the overall complexity is $O(NN_c D^2 d)$. Note that the initialization phase (where

one calculates the distance between the first patches) involves computing all values of the $L_K^2$ array, leading to a complexity of $O(N_c d^2)$. However this situation occurs only once, or to simplify the implementation only once per image column, so as long as $N_2 > d$ the initialization process has a complexity bounded by $O(NN_cD^2d)$.

A pseudo-code for this fast NLM-Pa is proposed in Algorithm 5 and more detail on the implementation is given in Subsection 5.2. Note that the chosen pseudo-code and implementation result from a compromise between efficiency and simplicity: by avoiding the initialization phase when $x$ moves to the next column by implementing a sum of shifted invariant columns, one would obtain a faster code (but without changing the overall complexity of $O(NN_cD^2d)$); by adapting the scan of $x$ and $y$ pixels so that the iteration next to $x$ and $y$ would be $x + l$ and $y + l$, one would reduce the memory size needed to record the $L_K^2$ array.

# 4 Experiments on Databases: Parameters Estimation and Experimental Results

Having fast implementations, the NLM usability is now determined by the ability to set its 3 (NLM-P) or 4 (NLM-Pa) parameters in order to obtain an effective denoising. Methods that propose to automatically adjust NLM parameters according to local or global characteristics of the image will not be considered here, insofar as they lead to algorithms that differ from the original NLM. The reader interested in such NLM extensions may consult [15, 9] and references therein. To estimate NLM parameters, the optimal values are calculated from a set of noise-free natural images on which is added on each plane a Gaussian noise with mean 0 and given standard deviation, so that input images may be deemed to satisfy the noisy image model (1) with a known $\sigma$.

## 4.1 Parameters Estimation Using Two Image Databases

To build the image database, the choice was to take the 21 color images proposed in the on-line demo of the IPOL article [4]. These images are assumed to be almost free of noise, as they have been obtained by taking a good quality photograph in broad daylight and they have been post-processed with a low-pass filter followed by a sub-sampling of factor 8. These color images contain 3 color planes in the RGB color model, each channel being coded using 8 bits. In the following, the corresponding database will be called *Color IPOL*.

Since the patch distances (11) and (9) are computed using all color components, the estimation of the oracle $|u(x) - u(y)|$ will be more reliable as the number of channels $N_c$ increases. Therefore, parameter's values (especially the patch side length $d$) should depend on the number of channels. To assess this effect, another database made of gray-level images (one 8-bits channel) is used. Such images are simply the grayed version of Color IPOL ones, using the Rec. 601 Luma weights[3]. The resulting database will be called *Gray-level IPOL*.

From any of these two databases, the procedure used to get optimal parameters is as follows. The standard deviation is quantified between 1 and 100 with a step of one. Gaussian noise with mean 0 and standard deviation $\sigma \in [\![1, 100]\!]$ is added to each image in the database, using the Mersenne Twister pseudo-random number generator provided by IPOL development tools[4]. In order to limit the impact of the noise realization, 10 noisy images per noise-free image are generated. This results in 210 different noisy images per database that are used as input of the optimization process. The function $f_\sigma$ to be maximized is the arithmetic mean of the Peak Signal to Noise Ratio (PSNR)

[3] http://en.wikipedia.org/wiki/Luma_%28video%29.
[4] https://tools.ipol.im/wiki/doc/tools.

**Algorithm 5:** Pseudo-code for a fast NLM-Pa: sums of invariant lines (SIL) algorithm

**input** : `V`, `ds`, `Ds`, `h`, `a`
**output**: `Vd`
**** INITIALIZATION ****
(`N1`, `N2`) ← image size
`Nc` ← number of color channels
`Vsym` ← symmetrized noisy image V with border Ds+ds
**if** $a > 0$ **then** *Kernel for NLM-Pa*
  `K` ← 1D-kernel of Gaussian Euclidean norm, computed using (20)
**else** *Kernel for NLM-P*
  `K` ← constant kernel $1/d$ for $d = 2 \times \texttt{ds} + 1$
**** MAIN LOOP *** denoise pixel $x = (x_1, x_2)$, center of the 1st patch*
**for** $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
  *— FIRST PASS — compute NLM-weights*
  **for** $y = (y1, y2) = (x1 - Ds, x2 - Ds)$ *to* $(x1 + Ds, x2 + Ds)$ **do**
    *$y = (y_1, y_2)$ is the center of the 2d patch*
    *Compute distance between the two patches using sums of invariant lines*
    $t \leftarrow y - x$; Dist2 ← 0
    **if** $x2 = 0$ **then** *$V(x)$ on upper side: compute and record $L^2_K$ for all lines following (22)*
      **for** $z2 = -ds$ *to* $+ds$ **do**
        l2 ← 0
        **for** $c = 0$ *to* $Nc - 1$ **do**
          **for** $z1 = -ds$ *to* $+ds$ **do**
            $\text{l2} \leftarrow \text{l2} + K(z1) \times (\texttt{Vsym}[c](x + z) - \texttt{Vsym}[c](y + z))^2$
        $L2(x, t, z2) \leftarrow l2$  *See Subsection 5.2 for details on L2 array*
        Dist2 ← Dist2 + $K(z2) \times l2$
    **else** *Use previously computed $L^2_K$ with a shift and compute last line following (24)*
      **for** $z2 = -ds$ *to* $ds - 1$ **do**
        Dist2 ← Dist2 + $K(z2) \times L2(x - (0, 1), t, z2 + 1)$  *Sum of invariant lines, see 5.2*
      l2 ← 0
      **for** $c = 0$ *to* $Nc - 1$ **do**
        **for** $z1 = -ds$ *to* $+ds$ **do**
          $\text{l2} \leftarrow \text{l2} + K(z1) \times (\texttt{Vsym}[c](x + z) - \texttt{Vsym}[c](y + z))^2$
      $L2(x, t, z2) \leftarrow l2$
      Dist2 ← Dist2 + $K(z2) \times l2$
    *Compute unnormalized weight $w(x, y)$ following (5) and (12)*
    $W(x, y) = e^{-\text{Dist2}/(Nc \times h^2)}$
  *— SECOND PASS — compute denoised pixel $\tilde{u}(x)$*
  **for** $c = 0$ *to* $Nc - 1$ **do**
    $r \leftarrow 0$  *Sum of weighted pixel's values, without normalization*
    $s \leftarrow 0$  *Sum of weights, for normalization*
    **for** $y = (y1, y2) = (x1 - Ds, x2 - Ds)$ *to* $(x1 + Ds, x2 + Ds)$ **do**
      $r \leftarrow r + W(x, y) \times \texttt{Vsym}[c](y)$  *The weighted average is done here*
      $s \leftarrow s + W(x, y)$  *Compute sum of weights, for normalization*
    *Final estimate based on the assumption that original planes take values in $[0, 255]$*
    $\texttt{Vd}[c](x) \leftarrow \min(\max(r/s, 0), 255)$

between each noisy image and its noise-free version and the optimization problem may be written, in the case of the NLM-Pa denoising scheme and for $\mathcal{B}$ the image database,

$$f_\sigma^{\text{NLM-Pa},\mathcal{B}}(d_s^\star, D_s^\star, h^\star, a^\star) = \max_{d_s,D_s,h,a} \left\{ \frac{1}{10|\mathcal{B}|} \sum_{u\in\mathcal{B}} \sum_{i=1}^{10} \text{PSNR}(u, \tilde{u}_{\text{NLM-Pa}}(u,i,\sigma,d_s,D_s,h,a)) \right\}, \quad (26)$$

where $\tilde{u}_{\text{NLM-Pa}}(u,i,\sigma,d_s,D_s,h,a)$ is the denoised image computed from the noisy input $u+\epsilon_i$ ($\epsilon_i$ between a noise realization of standard deviation $\sigma$), using NLM-Pa algorithm with parameters $d_s, D_s, h, a$. The same writing is obtained for $f_\sigma^{\text{NLM-P},\mathcal{B}}$, without the last parameter $a$.

Since the complexity of $f_\sigma$ does not allow the use of conventional optimization algorithms, the only safe method is to discretize the parameters and try as many combinations as possible. Fortunately, parameters $d_s$ and $D_s$ are integers and they can reasonably be limited to small bounds. As mentioned in the literature (see e.g. [1, 4]), the parameter $h$ that controls the decay of the weights should be proportional to the value of $\sigma$ and in practice, a value of $h$ slightly greater than $\sigma$ seems to be the best. The literature provides little guidance on the choice of the parameter $a$. In [11] (`nlmeans` module), the default value $a = d_s/2$ is proposed. Based on these considerations, Table 2 gives the quantized values and their bounds that have been used for the parameter exploration. Exploring all of these parameters would require 329120 calls to the NLM-Pa code for a given input image and standard deviation, so the parameters estimation for the Color or the Gray-level IPOL database would need more than $6.9 \times 10^9$ calls. Knowing that a call typically lasts a few seconds, it would not be possible to explore all the 329120 parameters combination. To reduce the dimensionality of the problem, a local optimum is sought with the following coordinate ascent algorithm where one cyclically iterates through each parameter direction:

$$\begin{cases} d_s^{k+1} = \arg\max_{d_s} f_\sigma^{\text{NLM-Pa},\mathcal{B}}(d_s, D_s^k, h^k, a^k) \\ D_s^{k+1} = \arg\max_{D_s} f_\sigma^{\text{NLM-Pa},\mathcal{B}}(d_s^{k+1}, D_s, h^k, a^k) \\ h^{k+1} = \arg\max_{h} f_\sigma^{\text{NLM-Pa},\mathcal{B}}(d_s^{k+1}, D_s^{k+1}, h, a^k) \\ a^{k+1} = \arg\max_{a} f_\sigma^{\text{NLM-Pa},\mathcal{B}}(d_s^{k+1}, D_s^{k+1}, h^{k+1}, a), \end{cases} \quad (27)$$

where $k$ is the current iteration. The algorithm stops when a stationary point is reached, i.e. when there is no improvement in the last cycle. In practice it appears that a stationary point is reached very rapidly, typically in 3 to 5 iterations. The results are given in Table 3 (NLMP-a) and Table 4 (NLM-P), where optimal values are piecewise-constant interpolated for non-integer values of $\sigma$ (when the expression does not show the $\sigma$ term).

| Parameter | Meaning | Quantized as integer | Range |
|---|---|---|---|
| $d_s$ | patch side half-length | $d_s$ | $d_s \in [\![1, 11]\!]$ |
| $D_s$ | search window side half-length | $D_s$ | $D_s \in [\![1, 17]\!]$ |
| $h = \sigma h_Q/10$ | filtering parameter, see (5) and (12) | $h_Q$ | $h_Q \in [\![5, 20]\!]$ |
| $a = a_Q/10$ | decay of the Gaussian kernel (10) | $a_Q$ | $a_Q \in [\![1, 110]\!]$ |

Table 2: Range of the NLM-P and NLM-Pa parameter exploration for the optimization process.

## 4.2 Experimental Results and Comparison With Blockwise NLM (NLM-B)

This subsection presents experimental results on NLM-P and NLM-Pa with previously computed optimal parameters, together with results obtained with the blockwise NLM (NLM-B) [1] (Section

Color IPOL database

| $\sigma$ | $d_s^\star$ | $D_s^\star$ | $h^\star$ | $a^\star$ |
|---|---|---|---|---|
| $]0,3]$ | 1 | 5 | $1.6\sigma$ | $(\sigma+2)/10$ |
| $]3,4]$ | 1 | 5 | $1.6\sigma$ | $(\sigma+1)/10$ |
| $]4,5]$ | 1 | 5 | $1.5\sigma$ | $(\sigma+1)/10$ |
| $]5,6]$ | 1 | 5 | $1.4\sigma$ | $(\sigma+1)/10$ |
| $]6,9]$ | 1 | 5 | $1.4\sigma$ | 0.7 |
| $]9,13]$ | 1 | 6 | $1.2\sigma$ | 1.0 |
| $]13,19]$ | 1 | 6 | $1.2\sigma$ | 1.1 |
| $]19,24]$ | 1 | 6 | $1.1\sigma$ | $\sigma/10$ |
| $]24,45]$ | 1 | 8 | $1.0\sigma$ | $\sigma/10$ |
| $]45,46]$ | 1 | 9 | $1.0\sigma$ | $\sigma/10$ |
| $]46,79]$ | 2 | 9 | $0.9\sigma$ | $\sigma/10$ |
| $]79,100]$ | 2 | 10 | $0.9\sigma$ | $\sigma/10$ |

Gray-level IPOL database

| $\sigma$ | $d_s^\star$ | $D_s^\star$ | $h^\star$ | $a^\star$ |
|---|---|---|---|---|
| $]0,1]$ | 3 | 3 | $1.7\sigma$ | 0.7 |
| $[1,3[$ | 3 | 3 | $1.7\sigma$ | 0.8 |
| $[3,4]$ | 3 | 3 | $1.7\sigma$ | 0.9 |
| $]4,5]$ | 3 | 3 | $1.7\sigma$ | 1.0 |
| $]5,7]$ | 3 | 4 | $1.6\sigma$ | 1.1 |
| $]7,9]$ | 3 | 4 | $1.4\sigma$ | 1.3 |
| $]9,13]$ | 3 | 5 | $1.3\sigma$ | 1.4 |
| $]13,18]$ | 3 | 5 | $1.3\sigma$ | 1.6 |
| $]18,19]$ | 3 | 5 | $1.3\sigma$ | 1.7 |
| $]19,20]$ | 3 | 5 | $1.2\sigma$ | $\sigma/10$ |
| $]20,28]$ | 3 | 6 | $1.1\sigma$ | $\sigma/10$ |
| $]28,67]$ | 3 | 7 | $1.0\sigma$ | $\sigma/10$ |
| $]67,83]$ | 3 | 8 | $1.0\sigma$ | $\sigma/10$ |
| $]83,100]$ | 4 | 8 | $1.0\sigma$ | $\sigma/10$ |

Table 3: Optimal parameters for NLM-Pa on Color and Gray-level IPOL databases.

Color IPOL database

| $\sigma$ | $d_s^\star$ | $D_s^\star$ | $h^\star$ |
|---|---|---|---|
| $]0,3]$ | 1 | 2 | $1.5\sigma$ |
| $]3,8]$ | 1 | 3 | $1.4\sigma$ |
| $]8,9]$ | 1 | 4 | $1.3\sigma$ |
| $]9,17]$ | 1 | 5 | $1.2\sigma$ |
| $]17,24]$ | 1 | 6 | $1.1\sigma$ |
| $]24,46]$ | 1 | 8 | $1.0\sigma$ |
| $]46,75]$ | 2 | 9 | $0.9\sigma$ |
| $]75,100]$ | 2 | 10 | $0.9\sigma$ |

Gray-level IPOL database

| $\sigma$ | $d_s^\star$ | $D_s^\star$ | $h^\star$ |
|---|---|---|---|
| $]0,7]$ | 1 | 3 | $1.5\sigma$ |
| $]7,9]$ | 1 | 4 | $1.4\sigma$ |
| $]9,19]$ | 1 | 5 | $1.3\sigma$ |
| $]20,28]$ | 2 | 6 | $1.1\sigma$ |
| $]28,47]$ | 3 | 6 | $1.0\sigma$ |
| $]47,70]$ | 3 | 7 | $1.0\sigma$ |
| $]70,87]$ | 3 | 8 | $1.0\sigma$ |
| $]87,100]$ | 4 | 8 | $1.0\sigma$ |

Table 4: Optimal parameters for NLM-P on Color and Gray-level IPOL databases.

5.5.2), also called patchwise NLM in the IPOL article [4]. The code used for NLM-B is the one published in [4] with the given parameters (see Gray and Color tables). The article does not mention how these parameters have been set. In NLM-B, the denoising process is performed by blocks using a vectorial NLM. From these restored overlapping blocks, the final estimate is computed by an aggregation formula. To know what to expect about comparison between pixelwise and blockwise NLM, let us quote [4]: "*The main difference of both versions is the gain on PSNR by the patchwise implementation, due to the larger noise reduction of the final aggregation process. Spurious noise oscillations near edges are also reduced by the final aggregation process. However, the overall quality in terms of preservation of details is not improved by the patchwise version.*"

All experiments were carried out by considering, as in Subsection 4.1, 10 noise realizations per noise-free image. Thus, results on Color and Gray-level databases are averages on 210 different denoised images. Figure 1 displays the corresponding curves for the Color IPOL database while Figure 2 is for the Gray-level IPOL database. In order to get a fair comparison, it is also necessary to run the NLM-P and NLM-Pa algorithms on another database containing a set of images disjoint to the IPOL set, on which parameters estimation has been performed. The *Volume 3: Miscellaneous*

set from the USC-SIPI Image Database [26] has been chosen because it includes some standard test images, such as House (record ID 4.1.05), Mandrill (4.2.03) and Lena (4.2.04). In what follows, the Miscellaneous Volume of the USC-SIPI Image Database is shortened by *USC-SIPI*. It contains 44 images (16 color and 28 monochrome) of very different nature and far away from the IPOL set, such as old photographs and test patterns. The PSNR curves (average on 440 denoised images) for the USC-SIPI database is given in Figure 3.

It is remarkable that PSNR curves on the USC-SIPI database are very similar to the ones with IPOL databases. This indicates that the computed optimal parameters are robust to image variation. Another indication confirming the relevance of these optimal parameters is in the convex and regular shape of the NLM-P and NLM-Pa curves. In contrast, the few irregularities that can be noted in NLM-B curves suggest that the parameters given in [4] are suboptimal in some standard deviation intervals, with respect to the PSNR criterion. One reason of the curves regularity is in the large PSNR averaging done on images of the database. However the curves regularity remains quite good on a single image, see Figure 4 where are plotted PSNR curves for the standard Lena image (record # 4.2.04 in the USC-SIPI database; remember that this image is not in the database used to compute optimal NLM-P and NLM-Pa parameters).

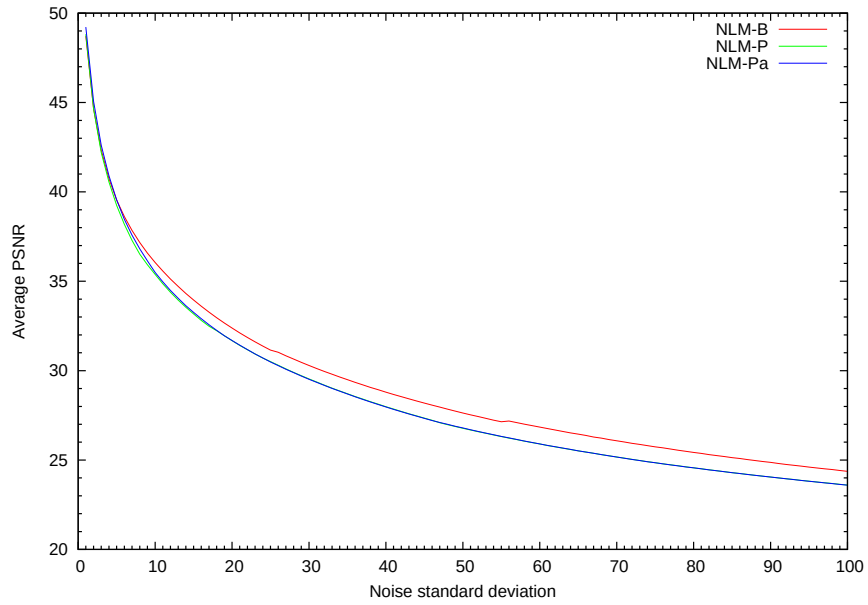


Figure 1: Average PSNR vs noise standard deviation for Color IPOL database.

Let us now compare the performance of the three denoising methods. The experiments confirm that the Gaussian Euclidean patch distance (9) does not bring much compared to the Euclidean one (11), see Figure 5 where average PSNR differences between NLM-Pa and NLM-P are drawn. The difference between NLM-B and NLM-Pa is plotted in Figure 6. In agreement with the text from [4] quoted above, NLM-B presents better PSNR than NLM-P/NLM-Pa when the noise standard deviation is greater than 3 (USC-SIPI), 5 (Color IPOL) or 15 (Gray-level IPOL). As expected the visual quality of the images is so similar that it is hard to notice any difference, see Figure 7 where the noise standard deviation is 20. At most we can remark that the NLM-P/NLM-Pa restored images appear a little bit more blurry that the NLM-B. By cons, a bit of plum on the back of the hat disappeared on the NLM-B denoised image while it is still discernible on NLM-P/NLM-Pa ones.

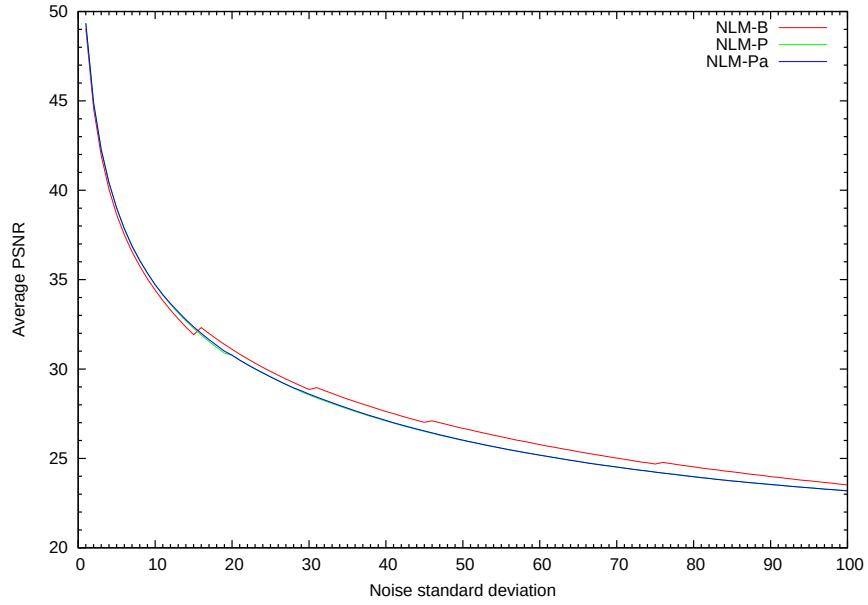


Figure 2: Average PSNR vs noise standard deviation for Gray-level IPOL database.

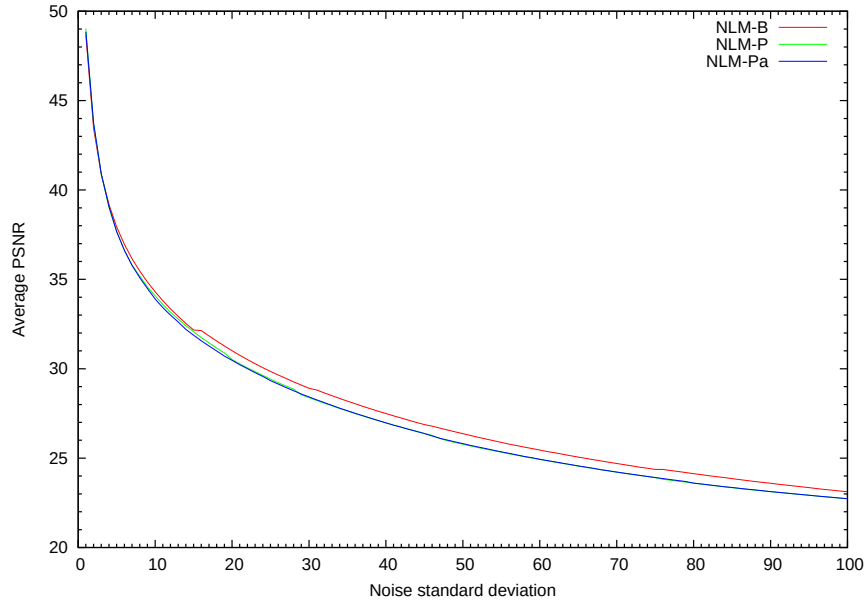


Figure 3: Average PSNR vs noise standard deviation for USC-SIPI database.

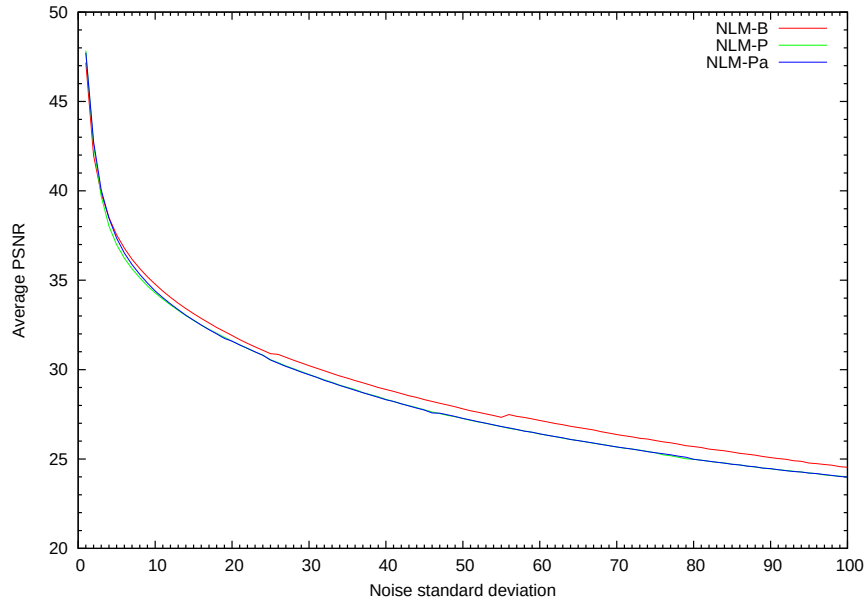


Figure 4: Average PSNR (over 10 noise realizations) vs noise standard deviation for the Lena image (record # 4.2.04 in USC-SIPI database).

Even for very high noise, where the PSNR difference between NLM-B and NLM-P/NLM-Pa is the largest, it remains difficult to notice any difference in visual quality, see Figure 8.

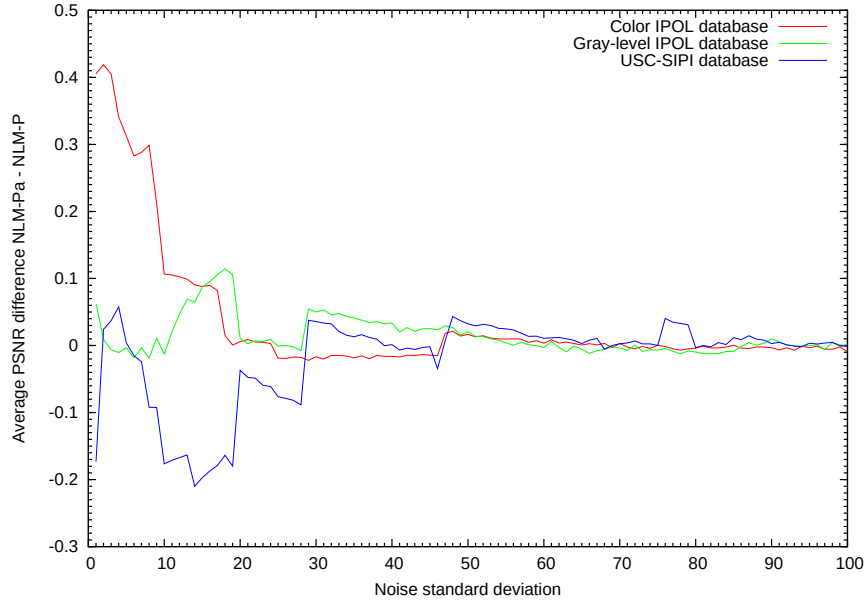


Figure 5: Average PSNR difference (NLMPa - NLMP) vs noise standard deviation.

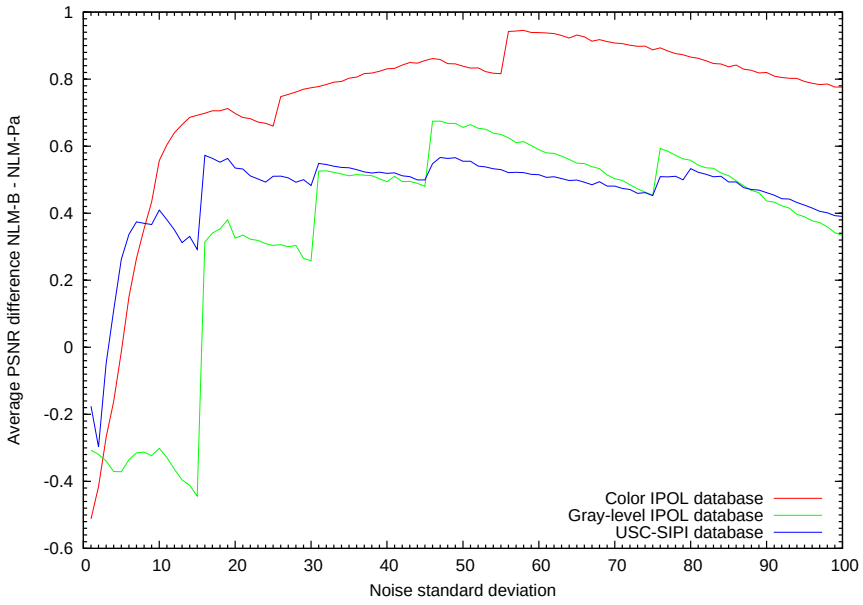


Figure 6: Average PSNR difference (NLM-B - NLMPa) vs noise standard deviation.

In order to compare the speed of NLM-B, NLM-P and NLM-Pa, we consider the total amount of CPU-time[5] spent by the denoising process alone (therefore excluding input/output to read and write images on disk as well as the generation of the noisy image). Note that this CPU-time does not represent the elapsed real time (wall clock) taken from the start of the process until the end: as all algorithms are implemented using parallel processing, on a multi-core computer the elapsed real time would be approximately the CPU-time divided by the number of used CPU (provided that no other program is running at the same time). Figure 9 gives the CPU-time spent on average on the Color IPOL database while Figure 10 is for the Gray-level IPOL database. Figure 11 displays the ratio of the CPU-time between NLM-B and NLM-Pa with the SIL algorithm. We first remark that on the Color IPOL database, the SIL algorithm is faster than the basic one in case of high noise only. The reason is simply in the optimum value of $d_s^\star = 1$ for $\sigma \leq 46$: for such a tiny patch it is useless to develop fast patch distance computation. The situation differs for NLM-Pa with grayscale images where $d_s^\star \geq 3$. Nevertheless, the increase in speed of the SIL algorithm compared to the basic one never exceeds the factor 4.5. However it should be noted that the speed increase may be much more important for NLM variants where the recommended patch size is greater than the maximum value of $d_s^\star = 4$ obtained here, see for example [14]. In the experiments the NLM-B algorithm appears much slower than NLM-P/NLM-Pa, with a CPU-time ratio between 6 (Color IPOL database, $\sigma = 25$) and 49 (Gray-level IPOL database, $\sigma = 83$). This may seem surprising in view of [1] (Section 5.5.2) where the blockwise NLM version is presented as a way to reduce the complexity of the pixelwise NLM. The reason is in the block overlapping that is not lowered in the NLM-B implementation [4]: using fewer blocks in the aggregation procedure would reduce the computation time, but probably also the denoising performance. Another element to explain the difference in speed between NLM-B and NLM-P/NLM-Pa is in the search window size, given by the parameter $D_s$. In the textbook case

[5]See `http://www.gnu.org/software/libc/manual/html_node/CPU-Time.html`. Processors used to perform the experiments were Intel Xeon L5640. All programs were compiled with OpenMP multithreading and all compiler optimizations turned on.

Figure 7: Visual comparison of parameter-free denoising algorithms applied on Lena image ($\sigma = 20$; zoom on a part but PSNRs are computed on the whole image; in this experiment the noise realization is the same for all of the three denoising algorithms). From left to right and top to bottom: original $u$; NLM-B (PSNR=31.90); NLM-P (PSNR=31.61); NLM-Pa (PSNR=31.61).

where $\sigma = 20$ and for the Gray-level IPOL database, the speed ratio reaches almost 18 while the patch size is larger for NLM-Pa ($d_s^\star = 3$ corresponds to a patch size of $7 \times 7$) than for NLM-B (patch size of $3 \times 3$). On the other hand, the search window size is only $11 \times 11$ for NLM-Pa where it is $21 \times 21$ for NLM-B. In fact, whatever the database and the noise standard deviation, the optimal search window size for NLM-P and NLM-Pa is always less than or equal to the one of NLM-B. Though one can not compare them precisely (the algorithms being slightly different), this small optimal size is

Figure 8: Visual comparison of parameter-free denoising algorithms applied on Lena image ($\sigma = 60$; zoom on a part but PSNRs are computed on the whole image; in this experiment the noise realization is the same for all of the three denoising algorithms). From left to right and top to bottom: original $u$; NLM-B (PSNR=27.18); NLM-P (PSNR=26.43); NLM-Pa (PSNR=26.43).

consistent with that given in [19] and the study confirms the common overestimation of the search window size. One may hope that the proposed fast pixelwise NLM implementation combined with optimal parameters giving a reasonable search window size will contribute to reconsider the original NLM scheme, usually considered as a slow denoising method.

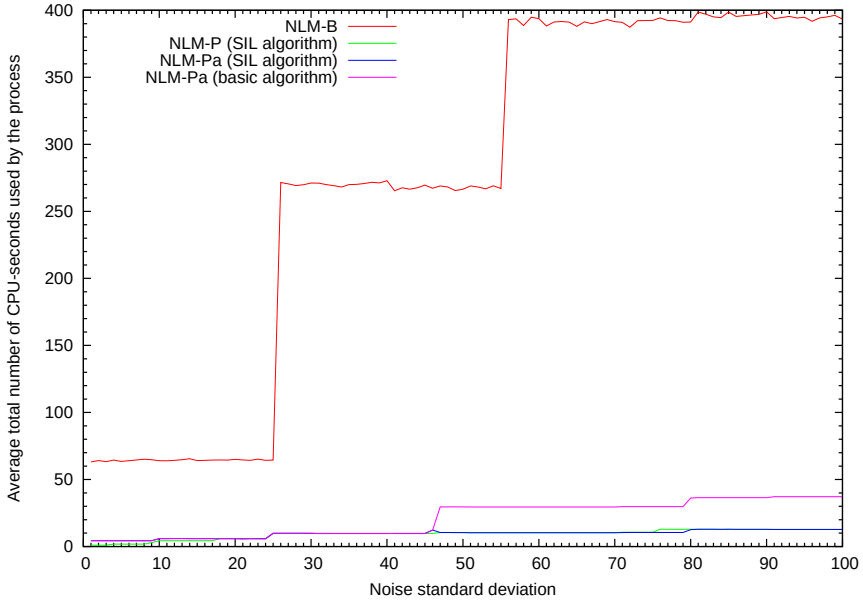


Figure 9: Total amount of CPU-time used by the denoising processes vs noise standard deviation: average over the Color IPOL database.

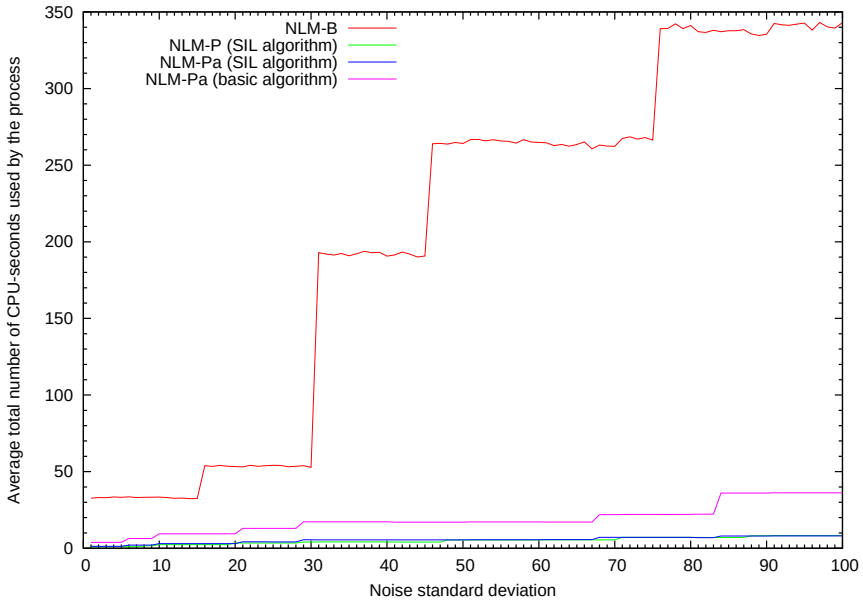


Figure 10: Total amount of CPU-time used by the denoising processes vs noise standard deviation: average over the Gray-level IPOL database.

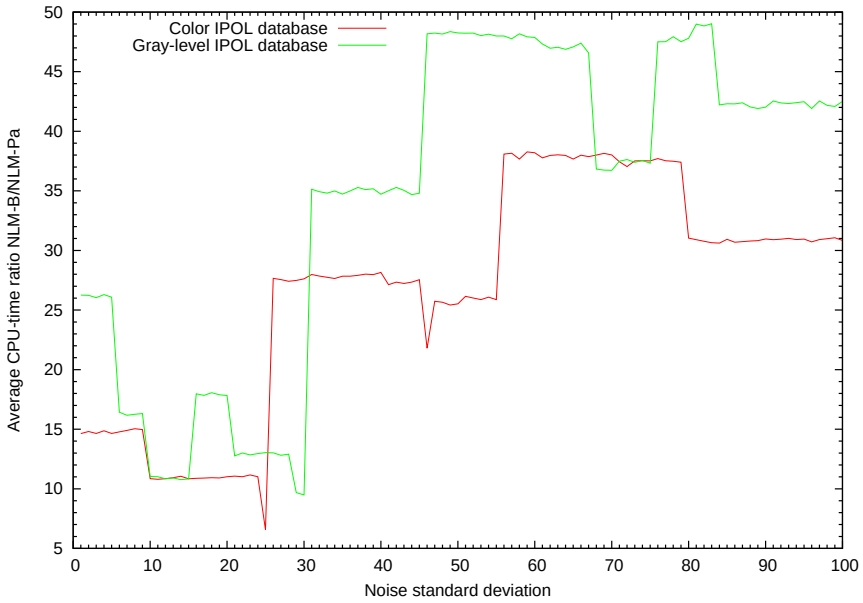


Figure 11: CPU-time ratio NLM-B/NLM-Pa (SIL algorithm) vs noise standard deviation for Color IPOL and Gray-level IPOL databases.

# 5 Source Code

The source code associated to this publication is available at the IPOL webpage[6]. It corresponds to an ANSI C (C89) implementation of Algorithm 1 and Algorithm 5. If available it uses the OpenMP 3.0 API to allow parallel computation. The main difference between the pseudo-code and the source code is in the way tables and in particular images are addressed. Indeed, the computer memory is a one-dimensional array of words and multidimensional arrays have to be addressed as one-dimensional ones. This is a very common constraint in image processing and the reader should not be confused by the necessary adaptation, including pointer manipulation to speed up array indexing. In the following is detailed the way the $L2$ array is addressed in the SIL algorithm, since it raises the question of which sum of lines has to be recorded.

## 5.1 Patent Warning

As the NLM denoising scheme may be covered by the European patent [2], part of the source code may use algorithms possibly linked to the patent. This source code is made available for the exclusive aim of serving as scientific tool to verify the soundness and completeness of the algorithm description. Compilation, execution and redistribution of the source code may violate exclusive patents rights in certain countries. The situation being different for every country and changing over time, it is the responsibility of the reader to determine which patent rights restrictions apply before compiling, using, modifying, or redistributing source code files.

[6] https://doi.org/10.5201/ipol.2014.120

## 5.2 Implementation of Fast NLM-Pa Using Sums of Invariant Lines

The pseudo-code of Algorithm 5 does not show how is managed the $L2$ array that records weighted quadratic distances between patches $V(x)$ and $V(x+t)$, denoted by $L^2_K(x,t,z_2)$. Two points are important to understand the chosen implementation: which samples are recorded in $L2$ and how the shift is performed along the line axis.

- The main loop is on pixels $x=(x_1,x_2)\in\Omega$ to be denoised and these pixels are scanned from top to bottom and left to right, so that the inner loop on line $x_2$ is done after the inner loop on column $x_1$. Thus, except when the pixel $x$ is on the first line of the image ($x_2=0$ case), its predecessor in the scan is $x-(0,1)$. Inside the main loop, values of $\{L^2_K(x,t,z_2)\}$ have to be recorded for $y=x+t\in\Omega_x$ and $z_2\in[\![-d_s,+d_s]\!]$ and one gets $D^2d$ samples. Computation of these values requires knowledge of $\{L^2_K(x-(0,1),t,z_2)\}$ for $y=x-(0,1)+t\in\Omega_{x-(0,1)}$ and $z_2\in[\![-d_s,+d_s]\!]$ and they were those that were calculated at the previous iteration $x-(0,1)$. Thus, only the parity of $x_2$ has to be indexed and the size of the $L2$ array is then of $2D^2d$ samples. The correspondence between $L2$ values and $L^2_K$ function values is the following: $L^2_K(x,t,z_2)=L2[(x_2\%2)\times D^2\times d+(y_2-x_2+D_s)\times D\times d+(y_1-x_1+D_s)\times d+z_2]$ where $\%$ is the integer modulo operator. Note that $y$ is not indexed by its absolute position in the plane $\Omega$ but by its relative position $t+(D_s,D_s)$ with respect to $x$.

- Looking more closely at how previously computed values of $L2$ at $x-(0,1)$ are used to compute values at $x$, equations (23) and (24) tell us that a shift of one $z_2$-line down has to be performed on the $L2$ array when one goes from a pixel $x$ to the next, for the same *relative* position of $y$. This can be easily done during the sum of invariant lines by updating the $L2$ array following Equation (23) that can be written in pseudo-code

$$L^2_K(x,t,z_2)\leftarrow L^2_K(x-l,t,z_2+1). \tag{28}$$

## 5.3 Lookup Table for Fast Computation of Exponential Function

Calculation of NLM-weights (12) and (5) from patches distance uses the computationally intensive exponential function $x\in\mathbb{R}^+\mapsto e^{-x}$. As it is not essential to get the NLM-weights with a high accuracy, one may accelerate the code using a Look-Up-Table (LUT). This LUT is an array that holds a set of precomputed values $(e^{-x_n})_n$, where the $(x_n)_n$ are uniformly sampled. When a particular value of $e^{-x}$ is wanted, the two closest values $e^{-x_n}$ and $e^{-x_{n+1}}$ ($x_n\le x\le x_{n+1}$) recorded in the LUT are read to estimate $e^{-x}$, using a linear interpolation. Such a LUT is part of the C++ code published in [4] to implement NLM-B and it is also used in the present C code to implement NLM-P/NLM-Pa. The precision of the estimation is given by the sampling period (that is, by the interval between two adjacent samples) and it has been tuned so that denoised images in the PNG output format (see below) would be identical, when using the original exponential function or the estimation with the LUT.

## 5.4 How Experiments Were Performed

The experiments reported in Section 4 were conducted using the reviewed C code as well as the C++ code published in [4], without any modification but the inclusion of functions needed to get and print the CPU-time. The optimization process and the presentation of results have required some additional code, written in Bash scripts[7] that are not included into this publication. As

[7] http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29.

the input/output images of the C and C++ codes are in PNG format[8], PSNR are computed on images with channel's values being integers (between 0 and 255) even though the denoising algorithm produces images with floating-point values. Therefore, results may slightly differ if simulations are redone by bypassing the PNG format. The alternative of a PSNR computation done before channel's values thresholding would have led to a result that does not accurately reflects the observed images.

# 6 Online Demo

An online demo running the source code is available at the IPOL webpage[9], with an interface similar to that of the NLM-B demo [4].

Inputs are the noise-free image $u$ (a grayscale or a 3-planes color image) and the standard deviation $\sigma$ of the added noise. The user may upload its own image or may choose any of the available images on the demo page. The algorithm may be applied on a subpart of the image, by clicking on two opposite corners in the displayed image. In order to keep a simple interface, the demo does not allow to pass options to the program. Therefore, the applied algorithm is always NLM-Pa with the SIL implementation (Algorithm 5). Outputs are the noisy image $v$ and the denoised one $\tilde{u}$. The PSNR as well as the RMSE (Root-Mean-Square Error) between $u$ and $\tilde{u}$ are displayed.

The user who wishes to select options, for example to run NLM-P or the basic implementation (Algorithm 1), should compile the source code (see Section 5) and run the program from its own computer.

# Acknowledgments

I thank the IRISA-UBS research center for allowing me access to their computer cluster partially funded by the CPER Invent'IST *masse de données* with the following partners: the French Ministry of Research, the Brittany Region, the General Council of Morbihan and the European Regional Development Fund.

# Image Credits


Standard test image (record # 4.2.04 in USC-SIPI image database [26]). In figures 7 and 8 the cropping $(200, 200) - (299, 299)$ has been performed.

# References

[1] A. Buades, B. Coll, and J.M. Morel, *A review of image denoising algorithms, with a new one*, Multiscale Modeling & Simulation, 4 (2005), pp. 490–530. http://dx.doi.org/10.1137/040616024.

[2] ——, *Image data processing method by reducing image noise, and camera integrating means for implementing said method*. ED Patent 1,749,278, February 2007.

[3] ——, *Image denoising methods. A new nonlocal principle*, Siam Review, 52 (1) (2010), pp. 113–147. http://dx.doi.org/10.1137/090773908.

[8] http://en.wikipedia.org/wiki/Portable_Network_Graphics.
[9] https://doi.org/10.5201/ipol.2014.120

[4] ——, *Non-local means denoising*, Image Processing On Line, (2011). http://dx.doi.org/10.5201/ipol.2011.bcm_nlm.

[5] P. Coupé, P. Yger, and C. Barillot, *Fast non local means denoising for 3D MR images*, in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2006, Springer, 2006, pp. 33–40. http://dx.doi.org/10.1007/11866763_5.

[6] F.C. Crow, *Summed-area tables for texture mapping*, ACM SIGGRAPH Computer Graphics, 18 (1984), pp. 207–212. http://dx.doi.org/10.1145/800031.808600.

[7] J. Darbon, A. Cunha, T.F. Chan, S. Osher, and G.J. Jensen, *Fast nonlocal filtering applied to electron cryomicroscopy*, in 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2008, pp. 1331–1334. http://dx.doi.org/10.1109/ISBI.2008.4541250.

[8] C.A. Deledalle, V. Duval, and J. Salmon, *Non-local methods with shape-adaptive patches (NLM-SAP)*, Journal of Mathematical Imaging and Vision, 43 (2012), pp. 103–120. http://dx.doi.org/10.1007/s10851-011-0294-y.

[9] V. Duval, J.F. Aujol, and Y. Gousseau, *A bias-variance approach for the non-local means*, SIAM Journal on Imaging Sciences, 4 (2011), pp. 760–788. http://dx.doi.org/10.1137/100790902.

[10] G. Facciolo, N. Limare, and E. Meinhardt, *Integral images for block matching*, Image Processing On Line, (2013). http://www.ipol.im/pub/pre/57/.

[11] J. Froment and L. Moisan (Eds), *Megawave2 v.3.01*. A free and open-source Unix image processing software for reproducible research, available at http://megawave.cmla.ens-cachan.fr, 2007.

[12] G. Gilboa and S. Osher, *Nonlocal linear image regularization and supervised segmentation*, Multiscale Modeling & Simulation, 6 (2007), pp. 595–630. http://dx.doi.org/10.1137/060669358.

[13] S. Grewenig, S. Zimmer, and J. Weickert, *Rotationally invariant similarity measures for nonlocal image denoising*, Journal of Visual Communication and Image Representation, 22 (2011), pp. 117–130. http://dx.doi.org/10.1016/j.jvcir.2010.11.001.

[14] Q. Jin, I. Grama, and Q. Liu, *Removing gaussian noise by optimization of weights in non-local means*. http://arxiv.org/abs/1109.5640, 2011.

[15] C. Kervrann and J. Boulanger, *Optimal spatial adaptation for patch-based image denoising*, IEEE Transactions On Image Processing, 15 (2006), pp. 2866–2878. http://dx.doi.org/10.1109/TIP.2006.877529.

[16] M. Lebrun, A. Buades, and J-M. Morel, *Implementation of the "Non-Local Bayes" (NL-Bayes) Image Denoising Algorithm*, Image Processing On Line, 2013 (2013), pp. 1–42. http://dx.doi.org/10.5201/ipol.2013.16.

[17] M. Mahmoudi and G. Sapiro, *Fast image and video denoising via nonlocal means of similar neighborhoods*, IEEE Signal Processing Letters, 12 (2005), pp. 839–842. http://dx.doi.org/10.1109/LSP.2005.859509.

[18] C. Pang, O.C. Au, J. Dai, W. Yang, and F. Zou, *A fast NL-means method in image denoising based on the similarity of spatially sampled pixels*, in IEEE International Workshop on Multimedia Signal Processing, 2009, pp. 1–4. http://dx.doi.org/10.1109/MMSP.2009.5293567.

[19] S. Postec, J. Froment, and B. Vedel, *Non-local means est un algorithme de débruitage local (Non-local means is a local image denoising algorithm)*, in Proceedings of GRETSI, France, 2013. http://arxiv.org/abs/1311.3768.

[20] B. Rajaei, *An analysis and improvement of the BLS-GSM denoising method*, Image Processing On Line, 4 (2014), p. 4470. http://dx.doi.org/10.5201/ipol.2014.86.

[21] J. Salmon, *On two parameters for denoising with non-local means*, Signal Processing Letters, 17 (2010), pp. 269–272. http://dx.doi.org/10.1109/LSP.2009.2038954.

[22] T. Tasdizen, *Principal neighborhoods dictionaries for nonlocal means image denoising*, IEEE Transactions on Image Processing, 18 (2009), pp. 2649–2660. http://dx.doi.org/10.1109/TIP.2009.2028259.

[23] R. Vignesh, B.T. Oh, and C-CJ Kuo, *Fast non-local means (NLM) computation with probabilistic early termination*, IEEE Signal Processing Letters, 17 (2010), pp. 277–280. http://dx.doi.org/10.1109/LSP.2009.2038956.

[24] P. Viola and M. Jones, *Rapid object detection using a boosted cascade of simple features*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, 2001, pp. I–511. http://dx.doi.org/10.1109/CVPR.2001.990517.

[25] J. Wang, Y. Guo, Y. Ying, Y. Liu, and Q. Peng, *Fast non-local algorithm for image denoising*, in IEEE International Conference on Image Processing, 2006, pp. 1429–1432. http://dx.doi.org/10.1109/ICIP.2006.312698.

[26] A.G Weber, *The USC-SIPI image database version 5*, tech. report, University of Southern California, 1997. http://sipi.usc.edu/database.

[27] I. Wegener, *Complexity Theory, exploring the limits of efficient algorithms*, Springer, 2005. ISBN-13 978-3540210450.