Published in Image Processing On Line on 2013–12–20.
Submitted on 2013–04–30, accepted on 2013–11–21.
ISSN 2105–1232 © 2013 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.2013.90

Analysis and Extension of the Percentile Method, Estimating a Noise Curve from a Single Image

Miguel Colom[1], Antoni Buades[2]

[1] Universitat de les Illes Balears, Spain and CMLA, ENS Cachan, France (miguel.colom@cmla.ens-cachan.fr)
[2] Universitat de les Illes Balears, Spain and CMLA, ENS Cachan, France (toni.buades@uib.es)

Communicated by Bartomeu Coll *Demo edited by* Miguel Colom

Abstract

Given a white Gaussian noise signal $\mathbf{N}_\sigma$ on a sampling grid, its variance σ^2 can be estimated from a small $w \times w$ pixels sample. However, in natural images we observe $\tilde{\mathbf{U}} = \mathbf{U} + \mathbf{N}_\sigma$, the combination of the geometry of the scene that is photographed and the added noise. In this case, estimating directly the standard deviation of the noise from $w \times w$ samples of $\tilde{\mathbf{U}}$ is not reliable since the measured standard deviation is not explained just by the noise but also by the geometry of $\mathbf{U}$. The Percentile method tries to estimate the standard deviation σ from $w \times w$ blocks of a high-passed version of $\tilde{\mathbf{U}}$ by a small p-percentile of these standard deviations. The idea behind is that edges and textures in a block of the image increase the observed standard deviation but they never make it decrease. Therefore, a small percentile (0.5%, for example) in the list of standard deviations of the blocks is less likely to be affected by the edges and textures than a higher percentile (50%, for example). The 0.5%-percentile is empirically proven to be adequate for most natural, medical and microscopy images. The Percentile method is adapted to deal with signal-dependent noise, which is realistic with the Poisson noise model obtained by a CCD device in a digital camera.

Source Code

The C++ implementation of the Percentile noise estimator has been peer reviewed and accepted by IPOL. The source code, the code documentation, and the online demo are available in the IPOL web page of this article[1].

Keywords: noise, noise estimation, percentile, Gaussian noise, homoscedastic noise, signal-dependent noise, DCT, noise curve

[1] https://doi.org/10.5201/ipol.2013.90

1 Introduction

Most digital images and movies are obtained by a CCD device and the main source of noise is the so-called *shot noise*. Shot noise is inherent to photon counting. The value $\tilde{\mathbf{U}}(\mathbf{i})$ observed by a sensor at each pixel $\mathbf{i}$ is a Poisson random variable whose mean would be the ideal image. The standard deviation of this Poisson distribution is equal to the square root of the number of incoming photons $\tilde{\mathbf{U}}(\mathbf{i})$ in the pixel captor $\mathbf{i}$ during the exposure time. This noise adds up to a thermal noise and to an electronic noise which are approximately additive and white.

For sufficiently large values of $\tilde{u}(\mathbf{i})$, $(\tilde{u}(\mathbf{i}) > 1000)$, the normal distribution $\mathcal{N}\left(\tilde{\mathbf{U}}(\mathbf{i}), \sqrt{\tilde{\mathbf{U}}(\mathbf{i})}\right)$ with mean $\tilde{\mathbf{U}}(\mathbf{i})$ and standard deviation $\sqrt{\tilde{\mathbf{U}}(\mathbf{i})}$ is an excellent approximation to the Poisson distribution. If $\tilde{\mathbf{U}}(\mathbf{i})$ is larger than 10, then the normal distribution still is a good approximation if an appropriate continuity correction is performed, namely $\mathbb{P}(\tilde{\mathbf{U}}(\mathbf{i}) \leq a) \simeq \mathbb{P}(\tilde{\mathbf{U}}(\mathbf{i}) \leq a + 0.5)$, where a is any non-negative integer.

As a rule of thumb, the noise model is relatively easy to estimate when the raw image comes directly from the imaging system, in which case the noise model is known and only a few parameters must be estimated. For this, efficient methods are described by Foi et al. [3, 4] for Poisson and Gaussian noise. The Percentile method [6] does not assume any model for the noise, with the exception that the variance of the noise depends only on the intensity of the ideal image, since an estimation that assumes that the noise is not signal-dependent is not realistic. Therefore, the output of the Percentile method is a noise curve, that is, a function that relates the intensity of the image to a noise standard deviation. The use of a small percentile makes the algorithm robust to the effect of textures and edges on the estimation of the noise variance. However, it might happen that for a certain intensity interval all the samples belong to textures and edges. In that case, the variance measured is explained by the geometry of the image and not the by noise. In order to minimize that effect, the noise curves are filtered as explained in section 3.2.

For a review of several noise estimation methods we refer the reader to the work of Lebrun at al. [6] and Olsen [10].

2 Noise Estimation Method

2.1 Notation and Terminology

This section prepares the detailed description of the noise estimation algorithm given in section 2.2 by fixing its notation and terminology.

- $\mathbf{U}$: the noiseless ideal image.
- $\tilde{\mathbf{U}}$: the noisy input image.
- N_x, N_y: the size of $\tilde{\mathbf{U}}$. N_x and N_y are always odd. If they are not, the leftmost column or the bottom row are first removed from $\tilde{\mathbf{U}}$.
- $\tilde{\mathbf{U}}_c$: the discrete noisy image $\tilde{\mathbf{U}}$ after cropping $(s-1)/2$ columns and rows from each of the four sides of $\tilde{\mathbf{U}}$ using the function $\mathrm{CROP}_{(s-1)/2}$, where s is odd. The details of this function are given in algorithm 1. The size of $\tilde{\mathbf{U}}_c$ is therefore $(N_x-(s-1))\times(N_y-(s-1)) = (N_x-s+1)\times(N_y-s+1)$.
- $\mathbf{R}$: the operator used to obtain the discrete filter $\mathbf{F}$ with support $s \times s$, with s odd.

- $\mathbf{F}$: the discrete filter with support $s \times s$ used to high-pass the noisy image, obtained from operator $\mathbf{R}$.
- $\mathbf{F}(x,y)$: the value of the discrete filter $\mathbf{F}$ at position $(x,y), x \in [0, s-1], y \in [0, s-1]$.
- $\wedge$: logical conjunction (*and* operator).
- $\vee$: logical disjunction (*or* operator).
- $\neg$: logical negation (*not* operator).
- $\tilde{\mathbf{U}^f}$: the cropped high-passed version of $\tilde{\mathbf{U}}$: $\tilde{\mathbf{U}^f} := \text{CROP}_{s-1}\left[(\tilde{\mathbf{U}} * \mathbf{F})\right]$, where $*$ is the discrete convolution operator:

$$\tilde{\mathbf{U}^f}(x,y) := \text{CROP}_{s-1}\left[(\tilde{\mathbf{U}} * \mathbf{F})(x,y)\right] = \text{CROP}_{s-1}\left[\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} \mathbf{F}_z(i,j)\tilde{\mathbf{U}}_z(x-i,y-j)\right] =$$

$$= \text{CROP}_{s-1}\left[\sum_{i=0}^{s-1}\sum_{j=0}^{s-1} \mathbf{F}(i,j)\tilde{\mathbf{U}}_z(x-i,y-j)\right], \tag{1}$$

where $\mathbf{F}_z(x,y) := \begin{cases} \mathbf{F}(x,y), & x \in [0,s-1] \wedge y \in [0,s-1] \\ 0 \quad \text{otherwise} \end{cases}$ extends $\mathbf{F}$ with zeros,

$\tilde{\mathbf{U}}_z(x,y) := \begin{cases} \tilde{\mathbf{U}}(x,y), & x \in [0,N_x-1] \wedge y \in [0,N_y-1] \\ 0 \quad \text{otherwise} \end{cases}$ extends $\tilde{\mathbf{U}}$ with zeros,

and the CROP_{s-1} function removes $s-1$ columns or rows at each of the four boundaries of the filtered image to avoid boundary effects. Since the convolution $(\tilde{\mathbf{U}} * \mathbf{F})(x,y)$ is defined for $x \in [0, N_x+s-2] \wedge y \in [0, N_x+s-2]$, the cropped image $\tilde{\mathbf{U}^f}(x,y)$ is defined for $x \in [0, N_x-s] \wedge y \in [0, N_y-s]$. The Fast Fourier Transform (FFT) [2] is used to speed-up the computation of the convolution; the details are given in section 2.2.1.

- $\tilde{\mathbf{U}}_c(x,y)$: the gray-level intensity of the pixel of $\tilde{\mathbf{U}}_c$ at column x and row y, $x \in [0, N_x-s]$, $y \in [0, N_y-s]$.
- $\tilde{\mathbf{U}^f}(x,y)$: the gray-level intensity of the pixel of $\tilde{\mathbf{U}^f}$ at column x and row y, $x \in [0, N_x-s]$, $y \in [0, N_y-s]$.
- w: the side of the overlapping $w \times w$ pixels blocks $\mathbf{W}(x,y)$.
- M: the total number of overlapping blocks. $M = (N_x-w-s+1)(N_y-w-s+1)$.
- $\mathbf{W}(x,y)$: a $w \times w$ pixels block in $\tilde{\mathbf{U}}_c$, $\mathbf{W}(x,y) = \{\tilde{\mathbf{U}}_c(x+i, y+j) : i \in [0,w-1], j \in [0,w-1]\}$.
- $\mathbf{W}^f(x,y)$: a $w \times w$ pixels block in $\tilde{\mathbf{U}^f}$, $\mathbf{W}^f(x,y) = \{\tilde{\mathbf{U}^f}(x+i, y+j) : i \in [0,w-1], j \in [0,w-1]\}$.
- $\mathbf{W}_m$: a $w \times w$ pixels block in $\tilde{\mathbf{U}}_c$ according to its index m in a list of overlapping blocks $\mathbf{W}_m = \mathbf{W}(x,y)$ where $y = \left\lfloor \frac{m}{N_x-s+1} \right\rfloor, x = m - y(N_x-s+1), m \in [0, M-1]$.
- $\mathbf{W}^f_m$: a $w \times w$ pixels block in $\tilde{\mathbf{U}^f}$ according to its index m in a list of overlapping blocks $\mathbf{W}^f_m = \mathbf{W}^f(x,y)$ where $y = \left\lfloor \frac{m}{N_x-s+1} \right\rfloor, x = m - y(N_x-s+1), m \in [0, M-1]$.

- $\mathbf{V}$: the list of M variances from the blocks $\{\mathbf{W}_m^f\}$. $\mathbf{V}[m]$ corresponds to the variance of the block $\mathbf{W}_m^f$.
- $\mathbf{T}$: the list of M means from the blocks $\{\mathbf{W}_m\}$. $\mathbf{T}[m]$ corresponds to the mean of the block $\mathbf{W}_m$.
- $\mathbf{I_V}[n]$: the index of the element at the position n in the list $\mathbf{V}$ after sorting the elements of $\mathbf{V}$ in ascending order.
- p: the (small) p-percentile.
- $\hat{\sigma}^2$: biased variance at the p-percentile of $\mathbf{V}$.
- $\tilde{\sigma}^2$: final unbiased variance at the p-percentile of $\mathbf{V}$.

2.2 The Algorithm

2.2.1 Step 1: Pre-filtering the Input Noisy Image

In order to get rid of deterministic tendencies due to signal structure, the image is first pre-filtered with a high-pass filter $\mathbf{F}$ implemented as a discrete stencil with support $s \times s$. This stencil corresponds to a discretization of a certain operator $\mathbf{R}$, typically a differential operator or a waveform. Convolving the image with such a filter removes smooth variations inside the blocks, which increases the number of blocks where noise dominates and on which the variance estimate will be reliable. Mastis proposed a similar approach [8], where operator $\mathbf{F}$ writes as a simple subtraction of the average or median to each 7×7 block. For the Percentile method a filter based on the DCT with support 7×7 is proposed. Given an $s \times s$ block in the image $\tilde{\mathbf{U}}$ at position (x, y), its orthonormal 2D DCT-II is

$$\mathrm{DCT}\left(\tilde{\mathbf{U}}(x,y)\right)(i,j) := Q_s(i)Q_s(j)\sum_{n_x=0}^{s-1}\sum_{n_y=0}^{s-1}\tilde{\mathbf{U}}(x+n_x, y+n_y)\cos\left[\frac{\pi}{s}\left(n_x+\frac{1}{2}\right)i\right]\cos\left[\frac{\pi}{s}\left(n_y+\frac{1}{2}\right)j\right] \tag{2}$$

with $x \in [0, N_x - s - 1], y \in [0, N_y - s - 1], i \in [0, s-1], j \in [0, s-1]$ and $Q_N(k)$ is the normalization factor

$$Q_N(k) := \begin{cases} \frac{1}{\sqrt{N}}, & k = 0 \\ \sqrt{\frac{2}{N}}, & k \neq 0. \end{cases} \tag{3}$$

Filter $\mathbf{F}$ is made by taking the normalized product of cosines that correspond to the highest frequency $[s-1, s-1]$, that is,

$$\mathbf{F}(n_x, n_y) := \frac{2}{s}\cos\left[\pi\left(n_x+\frac{1}{2}\right)\frac{s-1}{s}\right]\cos\left[\pi\left(n_y+\frac{1}{2}\right)\frac{s-1}{s}\right], \quad (n_x, n_y) \in [0, s-1]^2. \tag{4}$$

The filter $\mathbf{F}$ presented here was empirically proven to give the best results. However, other typical differential operators can be used, like directional derivatives, the Δ (Laplace) operator, or its iterations $\Delta\Delta$, $\Delta\Delta\Delta$, all implemented as discrete stencils. Figure 1 shows the discrete stencils associated to these operators.

The filtered image $\tilde{\mathbf{U}^f}$ is obtained by cropping the discrete convolution, $\mathrm{CROP}_{s-1}\left[(\tilde{\mathbf{U}} * \mathbf{F})(x,y)\right]$. Note that this cropping operation avoids the boundary effects of the convolution. To speed-up the computation, the Fast Fourier Transform (FFT) algorithm is used:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & -8 & 2 & 0 \\ 1 & -8 & 20 & -8 & 1 \\ 0 & 2 & -8 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & -12 & 3 & 0 & 0 \\ 0 & 3 & -24 & 57 & -24 & 3 & 0 \\ 1 & -12 & 57 & -112 & 57 & -12 & 1 \\ 0 & 3 & -24 & 57 & -24 & 3 & 0 \\ 0 & 0 & 3 & -12 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 1: From left to right, the discrete stencils associated with the discrete operators Δ, $\Delta\Delta$ and $\Delta\Delta\Delta$ discrete operators, respectively.

1. Consider the signal

$$\tilde{\mathbf{U}}_z(x,y) := \begin{cases} \tilde{\mathbf{U}}(x,y), & x \in [0, N_x - s] \wedge y \in [0, N_y - s] \\ 0 \quad \text{otherwise.} \end{cases} \tag{5}$$

   $\tilde{\mathbf{U}}_z(x,y)$ is defined for $x \in [0, N_x + s - 2] \wedge y \in [N_y + s - 2]$.

2. Consider the signal

$$\mathbf{F}_z(x,y) := \begin{cases} \mathbf{F}(x,y), & (x,y) \in [0, s-1]^2 \\ 0 \quad \text{otherwise.} \end{cases} \tag{6}$$

   $\mathbf{F}_z(x,y)$ is defined for $x \in [0, N_x + s - 2] \wedge y \in [N_y + s - 2]$.

3. Compute the FFT of $\tilde{\mathbf{U}}_z(x,y)$: $\text{FFT}[\tilde{\mathbf{U}}_z(x,y)]$.

4. Compute the FFT of $\mathbf{F}_z(x,y)$: $\text{FFT}[\mathbf{F}_z(x,y)]$.

5. Compute the point-wise product of the FFTs:

$$\text{FFT}[\tilde{\mathbf{U}}_z(x,y)] \times \text{FFT}[\mathbf{F}_z(x,y)].$$

6. Compute the inverse FFT of the point-wise product:

$$\text{FFT}^{-1}(\text{FFT}[\tilde{\mathbf{U}}_z(x,y)] \times \text{FFT}[\mathbf{F}_z(x,y)]).$$

7. Crop (see algorithm 1) the result to get $\tilde{\mathbf{U}^f}$, the cropped and low-pass filtered version of the noisy input image $\tilde{\mathbf{U}}$:

$$\tilde{\mathbf{U}^f}(x,y) := \text{CROP}_{s-1}\left[\text{FFT}^{-1}\left(\text{FFT}[\tilde{\mathbf{U}}_z(x,y)] \times \text{FFT}[\mathbf{F}_z(x,y)]\right)\right]. \tag{7}$$

Algorithm 1 Crops the boundary of the input image.

CROP$_b$ - Crops the boundary of the input image.
Input $\mathbf{I}$: input image.
Input N_x: width of $\mathbf{I}$.
Input N_y: height of $\mathbf{I}$.
Input b: width of the boundary that will be removed at each of the four sides of $\mathbf{I}$.
Output $\mathbf{V}$: the cropped image
$\mathbf{V} = \text{zeros}(N_x - 2b, N_y - 2b)$
for $y = b \dots N_y - b - 1$ **do**
    for $x = b \dots N_x - b - 1$ **do**
        $\mathbf{V}[x - b, y - b] = \mathbf{I}[x, y]$
    end for
end for

2.2.2 Step 2: Computing the Sample Variances from $\tilde{\mathbf{U}}^f$ and the Means from $\tilde{\mathbf{U}}$

The size of the filtered noisy image $\tilde{\mathbf{U}}^f$ is $N_x - s + 1 \times N_y - s + 1$ pixels. Therefore, there are $M = (N_x - w - s + 1)(N_y - w - s + 1)$ overlapping blocks of size $w \times w$ pixels. Each overlapping block is referred to as $\{\mathbf{W}_m^f\}$, where m is the index of the block, $m \in \{0, 1, \dots, M - 1\}$. Many noise estimation algorithms [5, 7, 9, 12] compute local estimates of the noise variance in small blocks that are used for a final statistical estimation (median, average, percentile, ...).

The empirical variance of each block $\mathbf{W}_m^f$ is computed as

$$\text{Var}\left(\mathbf{W}_m^f\right) := \frac{1}{w^2 - 1} \sum_{x=0}^{w-1} \sum_{y=0}^{w-1} [\mathbf{W}_m^f(x, y) - \bar{\mathbf{W}}_m^f]^2 \tag{8}$$

where $\bar{\mathbf{W}}_m^f = \frac{1}{w^2} \sum_{x=0}^{w-1} \sum_{y=0}^{w-1} \mathbf{W}_m^f(x, y)$ is the mean of the block. Let $\mathbf{V}$ be the list of variances of the blocks $\{\mathbf{W}_m^f\}$, $\mathbf{V}[m] := \text{Var}\left(\mathbf{W}_m^f\right)$. The corresponding means from $\{\mathbf{W}_m\}$ are stored in the list $\mathbf{T}[m] = \bar{\mathbf{W}}_m = \frac{1}{w^2} \sum_{x=0}^{w-1} \sum_{y=0}^{w-1} \mathbf{W}_m(x, y)$, $m \in \{0, 1, \dots, M - 1\}$. The mean of each patch will be needed when extending the method to signal-dependent noise (section 3.1), and therefore is stored at this stage of the method.

2.2.3 Step 3: Obtaining a (biased) Noise Variance Estimation from V Using p

Once the list $\mathbf{V}$ is built a biased noise variance estimation is obtained by the p-percentile. Set $\mathbf{I}_\mathbf{V}[n] := \text{SORTED}(\mathbf{V})[n]$ as the function that given a list of real numbers, sorts them in ascending order and returns the sorting indices. For example, if $n = 0$ then $\mathbf{I}_\mathbf{V}[0]$ is the index of the minimum in the list $\mathbf{V}$ and therefore $\mathbf{V}[\mathbf{I}_\mathbf{V}[0]]$ is that minimum. In this step, a biased estimation of the variance is obtained by the small p-percentile and is given by

$$\hat{\sigma}^2 = \mathbf{V}\left[\mathbf{I}_\mathbf{V}\left[\left\lfloor \frac{p}{100} M + \frac{1}{2} \right\rfloor\right]\right], \tag{9}$$

where M is the cardinal of $\mathbf{V}$. We obtain a list of variances $\mathbf{I}_\mathbf{V}[n]$ of $\mathbf{V}$ sorted in ascending order, $\mathbf{V}[\mathbf{I}_\mathbf{V}[n]]$ with $n \in [0, M - 1]$. Since in general the variance of the signal (geometry of the image) is higher than the variance of the noise, small percentiles of $\mathbf{V}$ are related more to the noise than to the signal. To illustrate it, figure 2 shows the noise-free test image *computer* after adding homoscedastic

Figure 2: Left: Noise-free test image *computer* (sand ee section 4.1) after adding homoscedastic white Gaussian noise of $\sigma = 10$. Right: pure homoscedastic white Gaussian noise of $\sigma = 10$ image. Both images have the same 704×469 size.

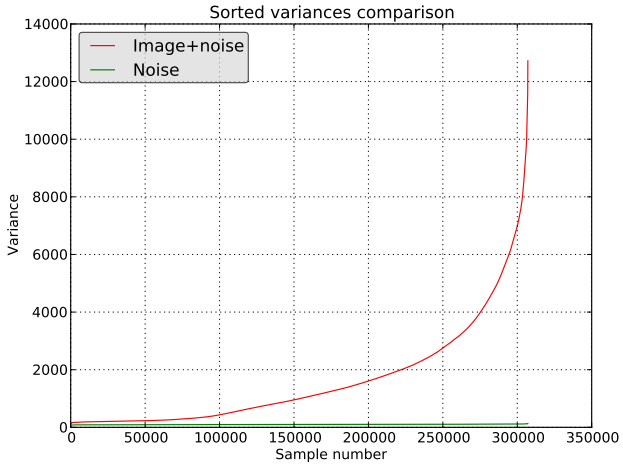


Figure 3: Values of $\mathbf{V}[\mathbf{I}_{\mathbf{V}}[n]]$ depending on n using 21×21 blocks without any filtering in the noise-free test image *computer* after adding homoscedastic white Gaussian noise of $\sigma = 10$ (red) and for pure homoscedastic white Gaussian noise of $\sigma = 10$ image (green). Only a part of the values is shown. The estimation in the *computer* image is only reliable when the percentile of $\mathbf{V}$ is small.

white Gaussian noise of $\sigma = 10$, and an image of pure homoscedastic Gaussian noise of $\sigma = 10$. Figure 3 shows the values of $\mathbf{V}[\mathbf{I}_{\mathbf{V}}[n]]$ depending on n and using 21×21 blocks without any filtering. We refer to section 4.1 for the details about how the noise-free images are obtained. Only for small percentiles the estimation of the variance on the *computer* image is close to the estimation on pure noise because of the effect of the image edges and textures. The Percentile method tries to avoid the effect of edges and textures by considering the variances under a very low percentile of the block variance histogram.

2.2.4 Step 4: Correcting the Biased Estimation $\hat{\sigma}^2$ to Obtain the Final $\tilde{\sigma}^2$ Estimation

When a percentile different from the median ($p = 0.5$) is used, the estimation of the variance obtained is biased by the percentile. Figure 4 shows the values of $\mathbf{V}[\mathbf{I}_{\mathbf{V}}[n]]$ depending on n using 21×21 blocks without any filtering in the image of pure white Gaussian noise (figure 2, right). If the percentile is under the median of the distribution an underestimation of the variance of the noise is obtained; on the other hand, if it is over the median, the result is an overestimation. The median is attained at position $n = \frac{(N_x - w + 1)(N_y - w + 1)}{2} = \frac{(704 - 21 + 1)(469 - 21 + 1)}{2} = 153558$. Since only a small percentile gives a

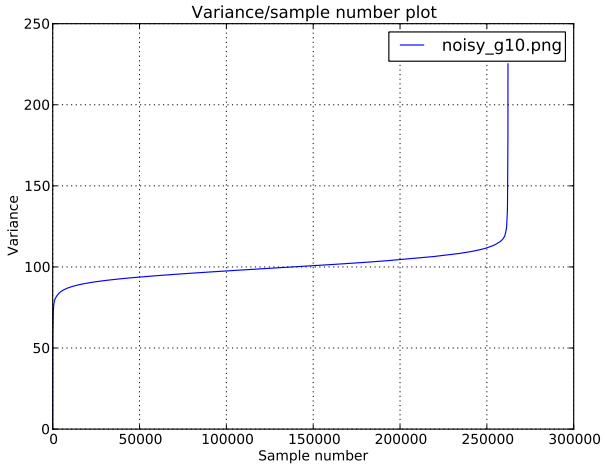


Figure 4: values of $\mathbf{V}[\mathbf{I}_{\mathbf{V}}[n]]$ depending on n using 21×21 blocks without any filtering in the image of pure white Gaussian noise (figure 2, right). If the percentile is under the median one obtains an underestimated value; if it is above the median, an overestimation. The median is attained at position $n = \frac{(N_x - w + 1)(N_y - w + 1)}{2} = \frac{(704 - 21 + 1)(469 - 21 + 1)}{2} = 153558$.

reliable biased estimation of the noise (see section 2.2.3), the estimation is lower than the real average block variance and it has to be corrected in order to get an estimation close to the median. The correction consists of multiplying the biased estimation $\hat{\sigma}$ by a factor. This correction only depends on the percentile, block size and on the chosen operator used to filter the image. Figure 5 shows the correction $\mathbf{C}_{w,\mathbf{R}}(p) = (\hat{\sigma} - \tilde{\sigma})_{w,\mathbf{R}}$ vs. the direct estimation $\hat{\sigma}$ learned with pure-noise images. The correction is linear with the observed $\hat{\sigma}$. As a matter of fact it can be easily proven that there exists a constant $k_{w,\mathbf{R}}$ such that $\hat{\sigma} = k_{w,\mathbf{R}}\mathbf{C}_{w,\mathbf{R}}(p) = k_{w,\mathbf{R}}(\hat{\sigma} - \tilde{\sigma})$ and then $(k_{w,\mathbf{R}} - 1)\hat{\sigma} = k_{w,\mathbf{R}}\tilde{\sigma}$. As a consequence,

$$\tilde{\sigma} = \frac{k_{w,\mathbf{R}} - 1}{k_{w,\mathbf{R}}}\hat{\sigma}. \tag{10}$$

Nevertheless, this constant $k_{w,\mathbf{R}}$ is not easy to calculate explicitly, but can be learned from simulations. To obtain it, a large image of 4320×3232 pixels with all pixels set to zero is used. Homoscedastic Gaussian noise with standard deviation σ is simulated and added to this image. Then, the noise is estimated from the noisy image using 200 bins, with a percentile $p \in \{0.01\%, 0.1\%, 0.5\%, 5\%, 10\%, 50\%\}$, a pre-filter operator $\mathbf{R}$, which can be chosen between the following: Identity (no filtering), Directional derivative, Laplace, Laplace (2 iterations), Laplace (3 iterations), Laplace (4 iterations), DCT with support 7×7, DCT with support 5×5, DCT with support 3×3 or the filter of the article Fast Noise Variance Estimation [5]. The size of the block is $w \times w$ with $w \in \{3, 7, 8, 21\}$. No curve filtering iterations are used. The averaged estimation along all the bins gives $\hat{\sigma}$.

The Fast Noise Variance Estimation method tries to avoid the influence of image structures (edges and textures) on the image when estimating the noise. To do it, it detects these structures using an operator based on the Laplacian and cancels them. It therefore considers two 3×3 Laplacian stencils L_1 and L_2 and computes their difference to obtain the noise estimation operator $L = 2(L_2 - L_1)$.

For example, with $p = 0.5\%$, $w = 21$ and $\mathbf{R} = \Delta\Delta\Delta$, this empirical $\frac{k_{w,\mathbf{R}}}{k_{w,\mathbf{R}}-1}$ factor learned on pure noise is 1.208610869.

The complete algorithmic description of the Percentile method is summarized in algorithm 2.

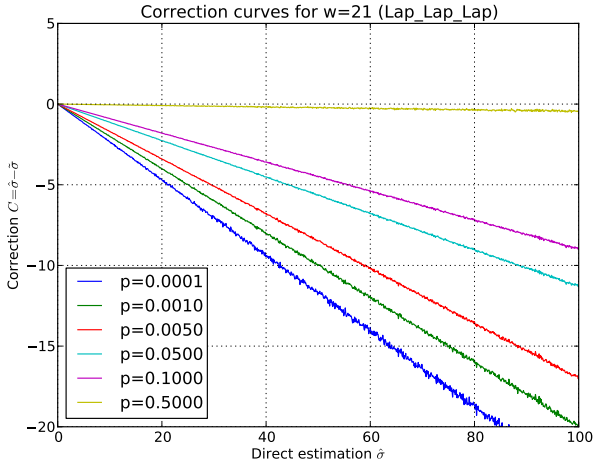


Figure 5: Corrections $\mathbf{C}_{w,\mathbf{R}}(p) = \hat{\sigma} - \tilde{\sigma}$ for several different percentiles, with 21×21 blocks and the $\mathbf{R} = \Delta\Delta\Delta$ operator, learnt on pure-noise patches.

Algorithm 2 Percentile noise estimation algorithm.

PERCENTILE - Returns the standard deviation of the image noise.
Input $\tilde{\mathbf{U}}$: discrete noisy image $\mathbf{U}$ after cropping.
Input N_x: (odd) width of the image before cropping.
Input N_y: (odd) height of the image before cropping.
Input $\mathbf{F}$: discrete filter with support $s \times s$.
Input s: support parameter of the discrete filter $\mathbf{F}$.
Input w: side of the $w \times w$ pixels blocks.
Output $\tilde{\sigma}$: estimated standard deviation of the image noise.
$\tilde{\mathbf{U}^f} = \text{CROP}_{s-1}\left[\tilde{\mathbf{U}} * \mathbf{F}\right]$. ▷ Filter $\tilde{\mathbf{U}}$ with the discrete filter $\mathbf{F}$ according to formula (1).
$M = (N_x - w - s + 1)(N_y - w - s + 1)$. ▷ Number of overlapping blocks.
for $m = 0 \dots M - 1$ **do**
    $\mathbf{W}^f(x,y) = \{\tilde{\mathbf{U}^f}(x+i, y+j) : i \in [x, x+w-1], j \in [y, y+w-1]\}$
    $\mathbf{W}^f_m = \mathbf{W}^f(x,y)$ where $y = \left\lfloor \frac{m}{N_x - s + 1} \right\rfloor, x = m - y(N_x - s + 1), m \in [0, M-1]$.
    $\mathbf{V}[m] = \text{Var}\left(\mathbf{W}^f_m\right) = \frac{1}{s^2-1} \sum_{x=0}^{s-1} \sum_{y=0}^{s-1} [\mathbf{W}^f_m(x,y) - \bar{\mathbf{W}^f_m}]^2$. ▷ Compute sample variances of the blocks.
end for
$\mathbf{I_V}[n] = \text{SORTED}(\mathbf{V})(n)\ \forall n \in [0, M-1]$. ▷ Get ascending sorting indices.
$\hat{\sigma}^2 = \mathbf{V}\left[\mathbf{I_V}\left[\left\lfloor \frac{p}{100} M + \frac{1}{2} \right\rfloor\right]\right]$. ▷ Get biased variance estimation.
$\hat{\sigma} = \sqrt{\hat{\sigma}^2}$. ▷ Get biased standard deviation.
$\tilde{\sigma} = \frac{k_{w,\mathbf{R}} - 1}{k_{w,\mathbf{R}}} \hat{\sigma}$. ▷ Obtain the final unbiased estimation by correcting $\hat{\sigma}$.

3 Extensions

3.1 Extension to Signal-Dependent Noise

Most noise estimation methods found in the literature [4, 5, 7, 9, 10, 11, 12] assume that the noise in the image is additive, signal-independent, and Poissonian–Gaussian. This assumption is not realistic

because of the physical nature of light and the way a CCD responds to light. It is well-known that the emission of photons by a body follows a Poisson distribution. This distribution can be approximated by a Gaussian distribution when the number of photons is large enough. For very dark regions of the image this assumption does not hold. Let us consider a pixel of the image, $\tilde{\mathbf{U}}(x, y)$, as a Poisson variable with variance and mean $\mathbf{U}(x, y)$. The Poisson noise has therefore a standard deviation of $\sqrt{\mathbf{U}(x, y)}$. An image is nothing but a noise whose mean would be the ideal image. This noise adds up to a thermal noise and to an electronic noise which are approximately additive and white. On a motionless scene with constant lighting, the expected value $\mathbf{U}$ can be approximated by simply accumulating photons for a long exposure time, and then by taking the temporal average of this photon count. Any noise estimation algorithm with the assumption of homoscedastic noise is unrealistic. Fortunately, most block-based methods can be easily adapted to signal-dependent noise.

For a signal dependent noise, a "noise curve" must be established. This noise curve associates with each image value $\mathbf{U}(x, y)$ an estimation of the standard deviation of the noise associated with this value. Thus, for each block in the image, its mean must be computed and will give an estimation of a value in $\mathbf{U}$. The measurement of the variation of the block (for example, its variance) will also be stored. The means are classified into a disjoint union of variable intervals or bins, in such a way that each interval contains a large enough number of elements. For the Percentile algorithm, the chosen minimum was 42000 elements/bin. These measurements allow for the construction of a list of block variances whose corresponding means belong to the given bin. Therefore, it is possible to apply the Percentile noise estimation algorithm to each set of blocks associated with a given bin. In this way, an estimation of the noise for the intensities inside the limits of the bin is obtained. Because the set of bins is disjoint and there is no gap between bins, it is possible to deduce by interpolation a curve that relates the means of the blocks with their standard deviation, hence obtaining a signal-dependent *noise curve*. To choose the intensity value associated with each bin, the median of the means of all blocks inside each bin is computed. The algorithmic description of the function building this histogram of block means can be found in algorithm 3. This algorithm works as follows:

1. It takes as input the number of bins that will be used ("bins" variable), the input data (the variances of the blocks, "data" variable), the associated intensities of the input data (the means of the blocks, "datal" variable) and the total number of samples ("N" variable). The algorithm stores at the variable "samples_per_bin" the integer value of N/bins. In general, samples_per_bin = 42000 samples/bin. Since the last bin contains the remaining samples, it may contain less than samples_per_bin samples.

2. It returns for each bin b its intensity bounds ("limits_begin[b]" and "limits_end[b]" variables), the list of variances that belong to bin b ("data_bins[b]" variable) and the list of intensities (block means) that belong to bin b ("datal_bins[b]" variable).

3. For each bin b, the algorithm fills the data_bins[b] and datal_bins[b] buffers with the variances and intensities of the blocks, sorted by their mean.

4. The lower and upper intensity bounds of the current bin b are stored into the variables limits_begin[b] and limits_end[b]. Then, the next bin is processed.

3.2 Filtering the Noise Curve

Optionally, the noise curve obtained on real images can be filtered. Indeed, it may present peaks when some given gray level interval contains mostly means of blocks belonging to a highly textured

Algorithm 3 Classifying blocks by their means.

CLASSIFY_BY_MEAN - Splits the input elements into disjoint *bins* according to the mean of the elements trying that each bin has the same cardinality.
Input bins: number of bins.
Input data: list of input data elements.
Input N: number of elements/bin.
Input datal: list of means of the input elements.
Output limits_begin[b]: the lower intensity bound for bin b.
Output limits_end[b]: the upper intensity bound for bin b.
Output data_bins[b]: list of elements at bin b.
Output datal_bins[b]: list of means of the elements at bin b.

```
samples_per_bin = ⌊N/bins⌋
limits_begin = zeros(bins)
limits_end = zeros(bins)
num_elements = zeros(bins)
data_bins = array(bins)
datal_bins = zeros(bins)
buffer = array(N)
bufferl = zeros(N)
indices = argsort(datal, N)                                  ▷ Sort data by datal
min_datal = datal[indices[0]]                                ▷ Min and max
max_datal = datal[indices[N-1]]
lim0 = min_datal
elements_count = 0
bin = 0
for idx = 0 ... N do
    if idx == N then
        finished_loading = true
    else
        lim1 = datal[indices[idx]]
        finished_loading = ¬ (bin == bins - 1) ∧ (elements_count ≥ samples_per_bin)
    end if
    if finished_loading then
        data_bins[bin] ← buffer
        datal_bins[bin] ← bufferl
        limits_begin[bin] = lim0          ▷ Update limits and number of elements of the bin
        limits_end[bin] = lim1
        num_elements[bin] = elements_count
        lim0 = lim1                                          ▷ Prepare for the next element
        bin = bin + 1
        elements_count = 0
    else
        buffer[elements_count] = data[indices[idx]]                  ▷ Keep loading...
        bufferl[elements_count] = datal[indices[idx]]
        elements_count = elements_count + 1
    end if
end for
limits_end[bins-1] = max_datal
```

region. In this case, the measured block variance would be caused by the signal itself and not by the noise and the noise variance would be overestimated.

Given the i-th control point of the noise curve $(\hat{\mu}_i, \hat{\sigma}_i)$ a closed intensity interval of radius D centered at this bin is considered, that is, $[\hat{\mu}_i - D, \hat{\mu}_i + D]$. For each intensity μ inside the interval (assuming that μ starts at $\hat{\mu}_i - D$ and is incremented with a step of 0.05 while it is less or equal to $\hat{\mu}_i + D$), consider the interpolated standard deviation that corresponds to each intensity μ. In order to avoid an excessive interpolation, if $\hat{\mu}_i - D < \hat{\mu}_0$ for the i-th bin, then the radius D is changed to the value $\hat{\mu}_i - \hat{\mu}_0$. In the same way, if $\hat{\mu}_i + D > \hat{\mu}_{B-1}$ (where B is the number of bins), then radius D is set to the value $\hat{\mu}_{B-1} - \hat{\mu}_i$. Since each $\hat{\mu}_i$ can be seen as an oscillation (given by the RMSE) around the ideal value, averaging the noise curve inside the interval $[\hat{\mu}_i - D, \hat{\mu}_i + D]$ for each control point attenuates the oscillations and puts them closer to the ground-truth. Once the oscillations have been attenuated, it might happen that a control point corresponds to a peak caused by a texture. In that case, the action taken is to compute the average inside the interval $[\hat{\mu}_i - D, \hat{\mu}_i + D]$ and to substitute the standard deviation $\hat{\mu}_i$ of the i-th control point by the average only if it is *lower* than the average of the intensities in the interval. In practice, this filtering procedure is iterated five times. In the first three iterations the control points are allowed to go up and down, thus canceling the oscillations around the ideal value. In the next two iterations the points are only allowed to go down, to attenuate the overestimation of the noise because of textures. The simple strategy presented here performs properly for most natural images and in general not more than five filtering iterations are needed to get a reliable estimation of the noise. The number of iterations was obtained empirically. Applying more than five iterations does not improve the results significantly and for certain images it could produce noise curves that are excessively smooth. The radius $D = 7$ was found empirically, too. The pseudo-code of the filtering is detailed in algorithm 6. It uses algorithm 5 to interpolate the standard deviation corresponding to a given intensity. Algorithm 5 uses algorithm 4 to get the corresponding standard deviation by a simple affine transformation.

Figure 6 shows the noise curve for the test image *Lena*. The non-filtered curve is drawn with solid lines and the filtered curve (five iterations) with dashed lines, using $D = 7$, $p = 0.5\%$, $w = 8$ and 6 bins. Note that the peak in the blue channel has decreased.

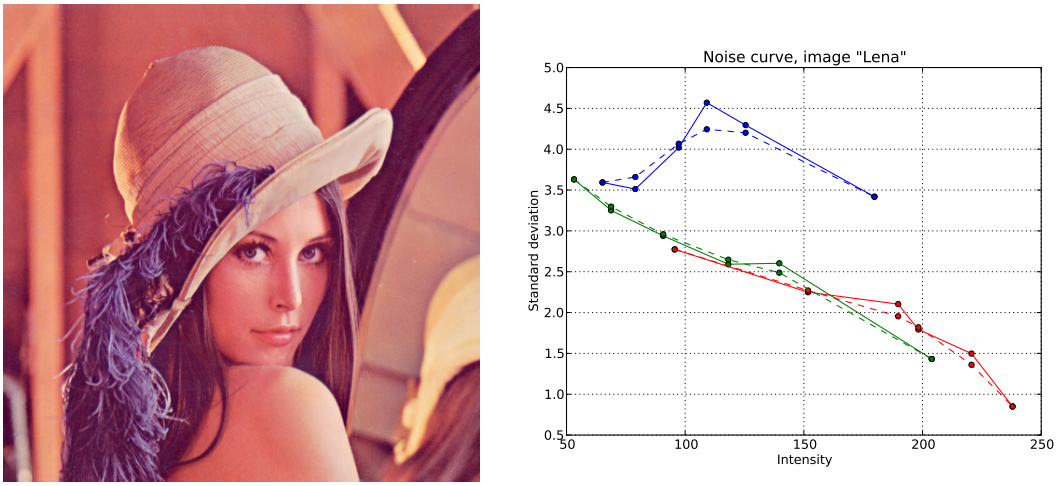


Figure 6: Left: test image *Lena* used to compare the noise curves with and without filtering. Right: noise curve for *Lena*. The red, green and blue plots correspond to the noise curves of each color channel. The non-filtered curve is drawn with solid lines and the filtered curve (five iterations) with dashed lines, using $D = 7$, $p = 0.5\%$, $w = 8$ and 6 bins. The peak in the blue channel is caused by textures of the image in that particular intensity range. After the filtering, the estimated standard deviation is lower.

Algorithm 4 Obtaining a corresponding standard deviation by an affine transformation.

AFFINE.
Input (μ_c, σ_c): current control point.
Input (μ_e, σ_e): endpoint control point.
Input μ: intensity of the control points whose standard deviation is wanted.
Output σ: standard deviation attributed to the intensity μ.

$\varepsilon = 10^{-6}$
if $|\mu_c - \mu_e| < \varepsilon$ **then**
    $s = 0$ ▷ Avoid dividing by zero
else
    $s = \frac{\sigma_c - \sigma_e}{\mu_c - \mu_e}$
end if
$\sigma = (\mu - \mu_e)s + \sigma_e$ **return** σ

Algorithm 5 Interpolates an affine standard deviation from of the points of the given noise curve.

INTERPOLATION.
Input (μ_c, σ_c): known control points.
Input μ: the intensity of the point whose interpolated standard deviation is wanted.
Output σ: the interpolated standard deviation of the point whose intensity is μ.

$i = \text{argmin}_i\,(\mu_c[i] - \mu|)$ ▷ Find the nearest control point
$m = \mu_c[i]$
if $\mu < m$ **then** ▷ on the right of μ
    if $i = 0$ **then**
        $i = 1$ ▷ Treat boundary
        $m = \mu_c[i]$
    end if
    $m_1 = \mu_c[i-1]$
    $m_2 = m$
    $s_1 = \sigma_c[i-1]$
    $s_2 = \sigma_c[i]$
else ▷ on the left of μ
    $N = \text{len}(\mu_c)$
    if $i \geq N - 1$ **then** ▷ Treat boundary
        $i = N - 2$
        $m = \mu_c[i]$
    end if
    $m_1 = m$
    $m_2 = \mu_c[i+1]$
    $s_1 = \sigma_c[i]$
    $s_2 = \sigma_c[i+1]$
end if
    return AFFINE$(m_1, s_1, m_2, s_2, \mu)$

Algorithm 6 Filters a noise curve.

FILTER_CURVE
Input (μ_c, σ_c): list of control points to be filtered.
Input D: radius.
Input allow_up: allow the points to go up and down. Otherwise, they are only allowed to go down.
Output σ_c^o: returned list filtered standard deviations
B = len(μ_c)
$\sigma_c^o \leftarrow \emptyset$
for $b = 0 \dots B-1$ **do**
    mu_current, std_current = $\mu_c[b], \sigma_c[b]$
    left = mu_current − D
    right = mu_current + D
    if left < $\mu_c[0]$ **then** ▷ Adjust the diameter for the points near the boundary
        dist = $\mu_c[b] - \mu_c[0]$
        left = mu_current - dist
        right = mu_current + dist
    else
        if right > $\mu_c[B-1]$ **then**
            dist = $\mu_c[B-1] - \mu_c[b]$
            left = mu_current - dist
            right = mu_current + dist
        end if
    end if
    sum_window = 0 ▷ Add the interpolated control points inside the interval [left, right]
    L = 0
    for μ = left ... right (with step $\Delta = 0.05$) **do**
        sum_window += INTERPOLATION(μ_c, σ_c, μ)
        L += 1
    end for
    std_new = sum_window / L
    if allow_up **then**
        std_filtered = std_new
    else
        std_filtered = std_new **if** std_new < std_current **else** std_current
    end if
    $\sigma_c^o \leftarrow$ std_filtered
end for
return σ_c^o

3.3 Discarding Saturated Pixels

When the number of photons measured by the CCD during the exposure time is too high, its output may get saturated, and therefore underestimated. When the signal saturates the output of the CCD, the measured variance in the saturated areas of the image is zero. Figure 7 shows an image with some saturated pixels. If the saturated pixels are taken into account when measuring the noise, the noise

Figure 7: Image "IMG_1071" with saturated pixels.

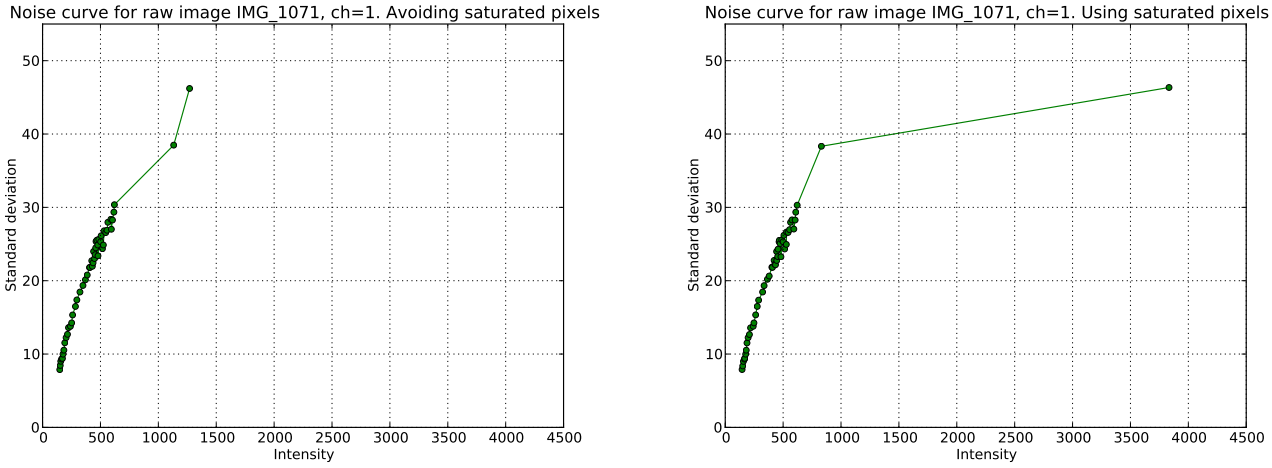
curve is not reliable anymore. Figure 8 shows a noise curve obtained from the image in figure 7 when the saturated pixels are avoided in the noise estimation (left) and when they are used (right). In this estimation 8×8 blocks and 49 bins were used. Since the intensity of the saturated pixels is much higher than the intensity of most pixels in the image, there is usually a large intensity gap between the values of normal non saturated pixels and those saturated (from about 600 to approximately 4000 in figure 8) since the over-exposed pixels represent usually outlier values. Even if there are few pixels with an intensity ranging between 600 and 4000, the noise curve will interpolate the standard deviation between the curve value at mean 600 and the noise curve value at mean 4000. Of course, the information given by the noise curve inside this gap is not correct at all. Therefore, it is better to detect and remove the saturated points before the noise curve estimation. In general, the strategy used to discard saturated pixels is to avoid the blocks that contain a group of four connected exactly equal pixels, in any of the channels. This is useful not only to discard saturated pixels, but also to avoid processing blocks whose pixels have suffered other types of alterations that can be detected by finding these special blocks. For example, it is well known that when encoding an image with the JPEG standard using a high compression factor, the coefficients at the high-frequencies of the 8×8 blocks of the image are set to zero or heavily quantized. This causes many undesired effects, like low frequency artifacts and blocking patterns. Undue smooth zones can also be created, because of the quantization of the high frequency coefficients.

In natural images that have not been deeply compressed, the probability of finding a set of four connected pixels sharing exactly the same value is very low, because of noise and textures. The pseudo-code and the details on how the pixels should be connected can be found in algorithm 7.

4 Evaluation of the Method

To evaluate the accuracy of the method, several kinds of tests were performed.

- Tests on simulated homoscedastic Gaussian noise using the images of figure 9. In this case



(a) Noise curve estimation avoiding the saturated pixels (b) Noise curve estimation affected by the saturated pixels

Figure 8: Noise curves obtained from the image in figure 7 when the saturated pixels are avoided in the noise estimation (a) and when they are taken into account (b). Using $w = 8$, $p = 0.5\%$ and 49 bins, blue channel.

Algorithm 7 Removal of equal pixels algorithm.

REMOVAL - Creates a mask of valid pixels.
Input I: input image.
Input N_x: width of **I**.
Input N_y: height of **I**.
Input w: block side.
Input num_channels: number of channels of **I**.
Output mask: mask of VALID/INVALID pixels.

$\varepsilon = 10^{-3}$
for $i = 0 \dots N_x - 1$ **do**
  for $j = 0 \dots N_y - 1$ **do**
    if $i < N_x - w + 1 \wedge j < Ny - w + 1$ **then** ▷ Check if the pixel is not too close to the image boundary
      for c = 0 ... num_channels - 1 **do**
        u = **I**.get_channel(c)
        pixel_status = (INVALID if c == 0 else mask[x,y])
        if $|u[i,j] - u[i+1,j]| > \varepsilon \vee |u[i+1,j] - u[i,j+1]| > \varepsilon \vee |u[i,j+1] - u[i+1,j+1]| > \varepsilon$ **then** ▷ Look if the 2×2 block is constant
          pixel_status = VALID ▷ Try to validate pixel
        end if
        mask[i, j] = pixel_status
      end for
    else
      mask[i, j] = INVALID
    end if
  end for
end for

we have taken seven and also one bins to classify the blocks according to their means (see section 3.1).

- Tests on a set of real raw images obtained by a Canon EOS 30D camera (see figure 10). The procedure explained in section 3.1 was used to get a noise curve. The results were compared to the ground-truth noise curve of the camera.

- Test on multiscale coherence. The standard deviation of a Gaussian white noise is divided by two when the image is down-scaled. By *down-scaling* the image we mean a sub-sampling of the image where each block of four pixels is substituted by their mean. This test checks if the measured noise is divided by two at each image after down-scaling.

4.1 Evaluation with Simulated Homoscedastic Noise

In this experiment, homoscedastic noise was simulated and then added to a set of ten noise-free images. Since the noise is perfectly known *a priori*, it can be used as a ground truth. One can therefore compute the RMSE of the standard deviation estimations. Figure 9 shows a set of 704×469 pixels noise-free images that were used in this test. In order to get the noise-free images, we have applied the following procedure. The pictures were taken with a Canon EOS 30D reflex camera of scenes under good lighting conditions and with a low ISO level. To reduce further the noise level, the average of each block of 5×5 pixels was computed, reducing therefore the noise by a factor of 5. Since the images are RGB, the mean of the three channels was computed, reducing the noise by a further $\sqrt{3}$ factor. Therefore the noise was reduced by a $5\sqrt{3} \simeq 8.66$ factor. Finally, the images, which already had a good SNR before they were processed, can be considered noise-free.

Figure 9: Set of noise-free images used to test the noise estimation algorithm with homoscedastic noise. From left to right and from top to bottom: bag, building1, computer, dice, flowers2, hose, lawn, leaves, stairs and traffic. Each image is 704×469 pixels.

To measure the error made when estimating the standard deviation σ of the simulated noise in the bin b in the image i, the RMSE along all the bins was used. This RMSE is denoted by $E^{(1)}_{i,\sigma}$ and it is defined by

$$E^{(1)}_{i,\sigma} := \sqrt{\frac{1}{|B|}\sum_{b=1}^{|B|} |\hat{\sigma}_{i,b} - \sigma|^2}, \tag{11}$$

where $|I|$ is the number of images, i is the image index ($1 \leq i \leq |I|$), $|B|$ is the number of bins, b is the index of the bin ($1 \leq b \leq |B|$), σ is the standard deviation of the simulated noise and $\hat{\sigma}_{i,b}$ is the estimated noise for the image i at the bin b. Table 1 shows the obtained $E^{(1)}_{i,\sigma}$ for each image i and

each σ of the simulated noise. A new image is added to the set of noise-free images, the *flat image*, which is a constant image where all pixels have the value 127. This permits to test the response of the algorithm to a pure white noise image. It is apparent that the highly textured images create a significant error, particularly when little noise was added. Estimates of noise below 2 are therefore obviously clearly unreliable. All in all, the estimate is nevertheless quite reliable for values $\sigma > 5$. Seven bins are used. The last row is the RMSE obtained for a given σ and all the images. It is denoted by $E_\sigma^{(2)}$ and defined as

$$E_\sigma^{(2)} := \sqrt{\frac{1}{|B||I|}\sum_{i=1}^{|I|}\sum_{b=1}^{|B|}|\hat{\sigma}_{i,b} - \sigma|^2} = \sqrt{\frac{1}{|I|}\sum_{i=1}^{|I|}\left(E_{i,\sigma}^{(1)}\right)^2}. \tag{12}$$

Image / $E_{i,\sigma}^{(1)}$	$\sigma = 1$	$\sigma = 2$	$\sigma = 5$	$\sigma = 10$	$\sigma = 20$	$\sigma = 50$	$\sigma = 80$
bag	0.74	0.62	0.47	0.37	0.81	0.73	2.26
building1	0.34	0.24	0.55	0.62	0.82	1.20	1.58
computer	0.35	0.36	0.55	0.64	0.86	1.25	3.10
dice	0.12	0.12	0.17	0.24	0.50	1.20	1.68
flowers2	0.15	0.13	0.15	0.27	0.81	1.54	2.82
hose	0.87	0.62	0.49	0.41	0.49	1.37	1.55
lawn	0.99	1.20	0.86	0.64	0.62	1.64	1.90
leaves	1.43	1.11	0.95	0.74	0.70	0.96	1.67
stairs	0.94	0.89	0.66	0.56	0.64	0.89	1.34
traffic	0.45	0.42	0.56	0.58	0.91	1.36	2.23
Flat image	0.02	0.04	0.15	0.10	0.20	1.25	2.63
$E_\sigma^{(2)}$	**0.58**	**0.52**	**0.51**	**0.47**	**0.67**	**1.22**	**2.07**

Table 1: This table shows the $E_{i,\sigma}^{(1)}$ RMSE after adding simulated noise to the set of noise-free images (figure 9) with several values of standard deviation σ. The last row is the $E_\sigma^{(2)}$ RMSE using the estimated $\hat{\sigma}_{i,b}$ of all the images. The parameters used in the experiments are percentile $p = 0.005$, block of size 15×15, DCT filter with support 7×7 and seven bins.

For completeness, the results corresponding to the estimation using just a single bin are shown in table 2, although this model of signal-independent noise using a single bin is not realistic at all.

Table 3 shows the obtained $E_\sigma^{(2)}$ RMSE depending on the number of the iterations of the noise curve filter (see section 3.2). Using five filtering iterations seems to be safe for any image with independence of the standard deviation of the noise, the kind of textures or the number and type of the edges the image may contain.

4.2 Evaluation Comparing the Noise Curve of the Raw Image with the Ground Truth

In this evaluation, the noise curve obtained by the algorithm for the raw images in figure 10 (12 bits/channel, ISO 1600, t=1/30s) was compared to the "ground truth" noise curve of the camera for that ISO. The ground truth was obtained by computing for each pixel the standard deviation of a large burst [1] of fixed snapshots of the same calibration pattern (figure 11).

To get the ground truth of the camera, we fixed the ISO sensitivity (in this case at ISO 1600) and used four exposure times, $t \in \{1/30s, 1/250s, 1/400s, 1/640s\}$. For each exposure time about

Image / $E^{(1)}_{i,\sigma}$	$\sigma = 1$	$\sigma = 2$	$\sigma = 5$	$\sigma = 10$	$\sigma = 20$	$\sigma = 50$	$\sigma = 80$
bag	0.71	0.60	0.43	0.32	0.65	0.29	1.22
building1	0.21	0.22	0.29	0.21	0.51	0.55	1.24
computer	0.29	0.25	0.38	0.32	0.08	0.99	0.86
dice	0.11	0.11	0.05	0.08	0.29	1.06	1.24
flowers2	0.13	0.10	0.12	0.03	0.00	0.54	2.03
hose	0.60	0.44	0.28	0.19	0.11	0.29	0.48
lawn	0.68	0.81	0.65	0.50	0.28	0.74	0.39
leaves	1.34	1.10	0.91	0.64	0.51	0.76	0.05
stairs	0.82	0.80	0.59	0.40	0.56	0.38	0.52
traffic	0.32	0.33	0.42	0.21	0.58	0.50	0.34
Flat image	0.02	0.03	0.05	0.05	0.16	0.61	1.93
$E^{(2)}_{\sigma}$	**0.47**	**0.44**	**0.38**	**0.27**	**0.34**	**0.61**	**0.94**

Table 2: This table shows the $E^{(1)}_{1,\sigma}$ RMSE after adding simulated noise to the set of noise-free images (figure 9) with several values of standard deviation σ. The last row is the $E^{(2)}_{\sigma}$ RMSE using the estimated $\hat{\sigma}_{1,b}$ of all the images. The parameters used in the experiment are percentile $p = 0.005$, block of size 15×15, DCT filter with support 7×7 and a single bin.

Image / $E^{(2)}_{\sigma}$	$\sigma = 1$	$\sigma = 2$	$\sigma = 5$	$\sigma = 10$	$\sigma = 20$	$\sigma = 50$	$\sigma = 80$
No filtering	0.58	0.52	0.51	0.47	0.67	1.22	2.07
1 iteration	0.57	0.52	0.49	0.44	0.60	1.12	1.84
2 iterations	0.56	0.52	0.49	0.42	0.57	1.07	1.75
3 iterations	0.56	0.51	0.48	0.41	0.55	1.04	1.70
4 iterations	0.55	0.50	0.47	0.39	0.53	1.02	1.68
5 iterations	0.55	0.50	0.46	0.38	0.52	1.00	1.67
6 iterations	0.54	0.49	0.46	0.38	0.51	0.99	1.66
7 iterations	0.54	0.48	0.45	0.37	0.50	0.98	1.65

Table 3: This table shows the obtained $E^{(2)}_{\sigma}$ RMSE values depending on the number of iterations of the noise curve filtering and the standard deviation of the noise (see section 3.2). The parameters used are percentile $p = 0.005$, block of size 15×15, the DCT filter with support 7×7 and seven bins. Five iterations is the recommend value, since using more iterations does not improve the result significantly and it could soften too much the noise curves for certain images.

two hundred pictures of the pattern were taken (see figure 11). After cropping the area of the image that does not contain the calibration pattern the final size of the raw image was 1352×1952. Since each 2×2 block of the CFA[2] contains one sample of the red channel, two samples of the green channel and one sample of the blue channel, one of the green channels can be discarded to get a single color pixel of each 2×2 block of the CFA, given an effective size of the color raw image of 676×976 pixels. Since the position of the camera was fixed when taking the snapshots of the image and assuming constant lighting, the variance along several samples coming from different images at the same pixel position could only be explained by the presence of noise. Therefore, it was possible to measure the mean of a block and the temporal standard deviation along all the snapshots to create an association mean→standard deviation, that is, a ground truth for camera noise curve, given the

[2]Color Filter Array.

Figure 10: Set of raw images used to test the noise estimation algorithm using 8×8 blocks, percentile 0.5%, 49 bins and without any noise curve filtering. The images are raw 12 bits/channel, taken with a Canon EOS 30D camera, ISO 1600 and exposure time t=1/30s. From left to right and up to bottom: images 1, 2, 3, 4, 5, 6, 7, and 8.

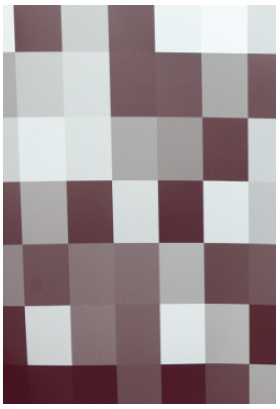


Figure 11: One of the pictures of the calibration pattern mire used to build the ground truth noise curve of the camera.

ISO and exposure times. Moreover, since the exposure time only affects the photon count and not the noise model, it was possible to overlap the noise curves for the four exposure times tested in a single ground truth noise curve depending only on the ISO parameter (see figure 12). Figure 13 shows an example of the noise curve obtained with the Percentile method. It matches with the ground truth quite accurately (see figure 12).

Given an estimated noise curve A of a test image, its control points are the pairs $(\hat{\mu}_{A,i,b}, \hat{\sigma}_{A,i,b}) \in A$ where $\hat{\mu}_{A,i,b}$ is the mean intensity for bin b and image i in A and $\hat{\sigma}_{A,i,b}$ is the corresponding standard deviation value for bin b and image i in A. In the same way, given a ground truth noise curve G, its control points are the pairs $(\mu_{G,v}, \sigma_{G,v}) \in G$. Unfortunately the means of the noise curve A and those in G do not necessarily coincide; that is, $\hat{\mu}_{A,i,b} \neq \mu_{G,v}$ for most (b, v) pairs. To solve this problem, instead of using G, a new ground truth curve $\tilde{G}_i$ is used. This $\tilde{G}_i$ curve has the same means $\hat{\mu}_{A,i,b}$ as A (and therefore the same number of bins), and its standard deviation values are obtained by a simple proportionality rule. Therefore, the control points in the new curve $\tilde{G}_i$ are $(\hat{\mu}_{A,i,b}, \tilde{\sigma}_{G_i,i,b}) = \left(\hat{\mu}_{A,i,b}, \frac{\sigma_{G,v+1}-\sigma_{G,v}}{\mu_{G,v+1}-\mu_{G,v}}(\hat{\mu}_{A,i,b} - \mu_{G,v+1}) + \sigma_{G,v}\right)$ where v is the index of the bin in the curve G such that $\mu_{G,v} \leq \hat{\mu}_{A,i,b} < \mu_{G,v+1}$ (see figure 14).

The error between the ground truth noise curve G and the test noise curve A for the image i and

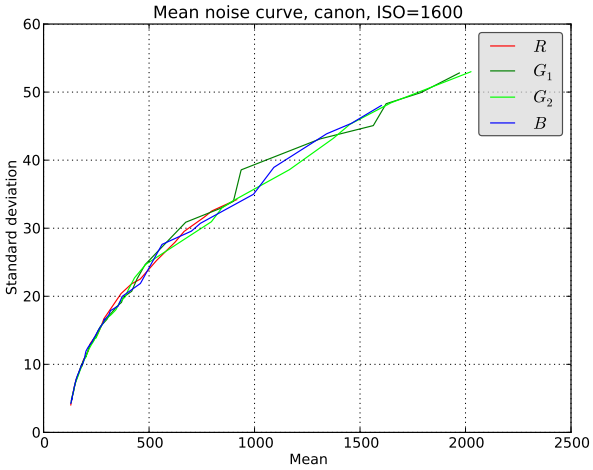


Figure 12: Ground truth of the Canon EOS 30D camera with ISO=1600.

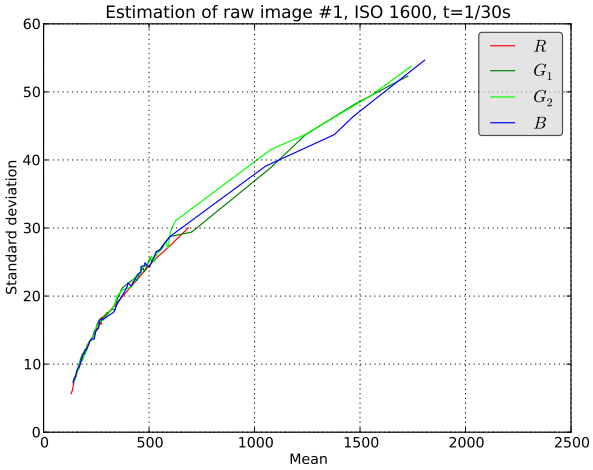


Figure 13: Noise curve obtained for the test image #1. Compare it with the ground truth noise curve in figure 12. Both curves match with small error.

bin b is defined as

$$E^{(3)}_{G,A,i,b} := |\tilde{\sigma}_{G_i,i,b} - \hat{\sigma}_{A,i,b}| = \left| \frac{(\sigma_{G,v+1} - \sigma_{G,v})(\hat{\mu}_{A,i,b} - \mu_{G,v+1})}{\mu_{G,v+1} - \mu_{G,v}} + \sigma_{G,v} - \hat{\sigma}_{A,i,b} \right|. \tag{13}$$

Img. 1	Img. 2	Img. 3	Img. 4	Img. 5	Img. 6	Img. 7	Img. 8
0.910	0.680	0.836	0.850	0.871	0.867	0.779	0.669

Table 4: Values of $E^{(3)}_{G,A,i,b}$ measuring the error between the noise curve A obtained for each test image i (figure 10) and the ground truth curve G for the Canon EOS 30D with ISO 1600. Note that the values of the intensities in a raw image are expressed in 12 bits and not with the usual 8 bits. Therefore, these errors should be divided by 16 in order to be compared with those obtained using 8 bits.

To test the average behavior of the algorithm in all the bins of any test image, we define a mean error function $E^{(4)}_{G,A,b}$ as the mean of the $E^{(3)}_{G,A,i,b}$ values over the $|I|$ images for each of the $|B|$ bins,

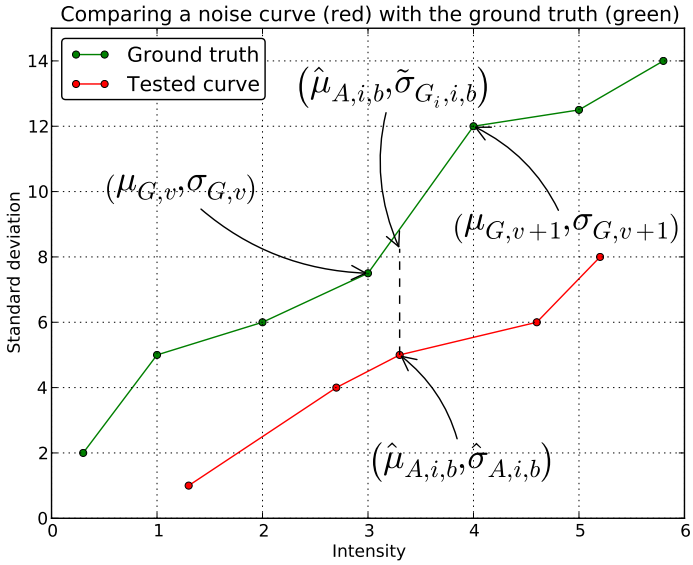


Figure 14: Checking a noise curve A (red) against the ground truth G (green), where i is the index of the image, b is the index of the bin, $(\hat{\mu}_{A,i,b}, \hat{\sigma}_{A,i,b})$ are the control points of the noise curve A, $(\mu_{G,v}, \sigma_{G,v})$ are the control points of G, and $\tilde{\sigma}_{G_i,i,b}$ is the standard deviation value projected from A into G.

that is,

$$E^{(4)}_{G,A,b} := \frac{1}{|I|} \sum_{i=0}^{|I|-1} E^{(3)}_{G,A,i,b}. \tag{14}$$

Figure 15 shows the error $E^{(4)}_{G,A,b}$ for all the 49 bins of the test images in figure 10 for the first green channel.

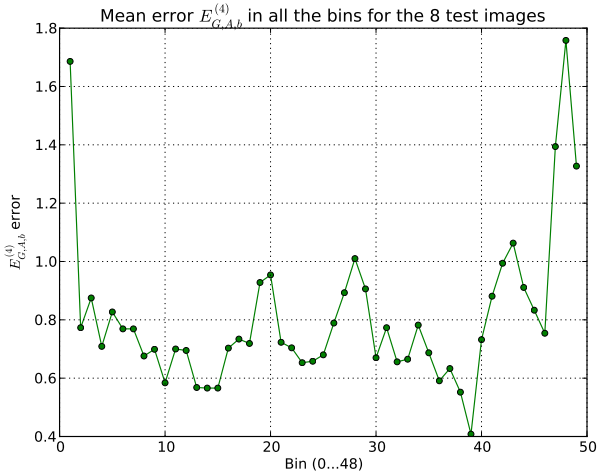


Figure 15: Mean error $E^{(4)}_{G,A,b}$ for the 49 bins of all the eight tests images in figure 10.

4.3 Evaluation of the Multiscale Coherence of the Result

Consider the down-scaling operator $\mathbf{S}$ that tessellates the image into 2×2 pixels blocks, and replaces each block by a pixel having the mean of the four previous pixels as new value. If $\tilde{\mathbf{U}}$ is a discrete pure Gaussian noise image with standard deviation σ, then $\mathbf{S}(\tilde{\mathbf{U}})$ has standard deviation $\frac{\sigma}{2}$. Indeed, if a block $\mathbf{W}$ contains the pixels $\{u_1, u_2, u_3, u_4\}$ each one with variance σ^2, the variance of the mean of $\mathbf{W}$ is $\mathrm{Var}\left(\bar{\mathbf{W}}\right) = \mathrm{Var}\left(\frac{u_1+u_2+u_3+u_4}{4}\right) = \frac{1}{16}[\mathrm{Var}\left(u_1\right) + \mathrm{Var}\left(u_2\right) + \mathrm{Var}\left(u_3\right) + \mathrm{Var}\left(u_4\right)] = \frac{1}{16}[4\sigma^2] = \frac{\sigma^2}{4}$. Therefore, the standard deviation is $\mathrm{Std}(\bar{\mathbf{W}}) = \frac{\sigma}{2}$; the noise has been divided by two. The objective of this test is to check if the noise estimation algorithm indeed divides the noise by two when the image is down-scaled several times.

$$\text{Set:} \quad E^{(5)}_{A_0,A_k,i,b} = \left| \frac{\tilde{\sigma}_{A_0,i,b}}{\hat{\sigma}_{A_k,i,b}} - 2^k \right| = \left| \frac{(\hat{\sigma}_{A_0,i,v+1} - \hat{\sigma}_{A_0,i,v})(\hat{\mu}_{A_k,i,b} - \hat{\mu}_{A_0,i,v+1})}{\hat{\sigma}_{A_k,i,b}(\hat{\mu}_{A_0,i,v+1} - \hat{\mu}_{A_0,i,v})} + \frac{\hat{\sigma}_{A_0,i,v}}{\hat{\sigma}_{A_k,i,b}} - 2^k \right|, \tag{15}$$

where

- A_k is the noise curve corresponding to the input image i after applying the down-scaling operator k times. For example, if $k = 2$ then A corresponds to the curve of the noise estimation of $\mathbf{SS}(\tilde{\mathbf{U}})$.
- i is the image index, for the raw images in figure 10, $1 \leq i \leq |I|$, where $|I|$ is the number of images. $|I| = 8$ images were used.
- b is the bin index, $1 \leq b \leq |B_k|$ where $|B_k|$ is the number of bins of the noise curve at scale k. For the test images $|B_0| = 49$, $|B_1| = 12$ and $|B_2| = 3$ bins are used.
- v is the index of the bin in the curve A_0 such that $\hat{\mu}_{A_0,i,v} \leq \hat{\mu}_{A_k,i,b} < \hat{\mu}_{A_0,i,v+1}$ (see figure 16).

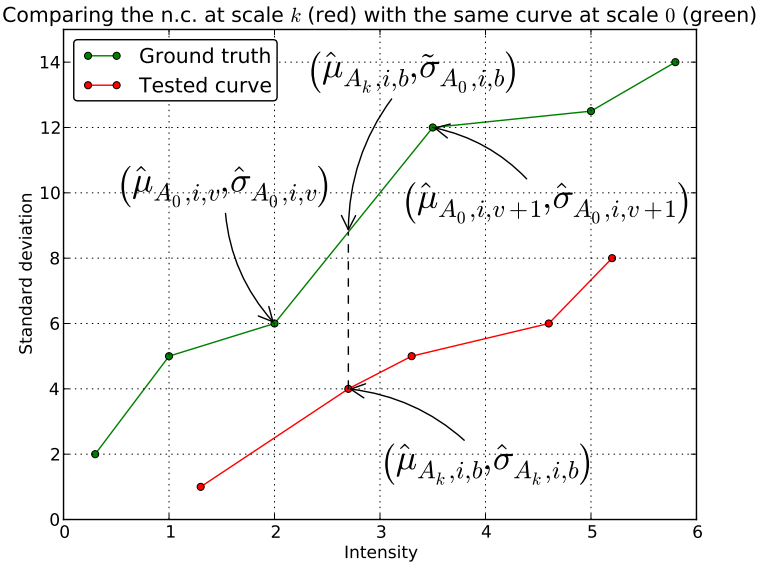


Figure 16: Checking a noise curve A_k at scale k (red) against the noise curve A_0 of the same image at scale 0 (green), where i is the index of the image, b is the index of the bin, $(\hat{\mu}_{A_k,i,b}, \hat{\sigma}_{A_k,i,b})$ are the control points of the noise curve of the sub-scaled image, $(\hat{\mu}_{A_0,i,v}, \hat{\sigma}_{A_0,i,v})$ are the control points of the noise curve of the image at scale 0, and $\tilde{\sigma}_{A_0,i,b}$ is the standard deviation value projected from A_k into A_0.

Remark: since the noise should be divided by two when operator $\mathbf{S}$ is applied and an ideal noise estimator is used, the relation $\frac{\tilde{\sigma}_{A_0,i,b}}{\hat{\sigma}_{A_k,i,b}}$ between the standard deviation estimations at scale 0 and scale k (applying k times $\mathbf{D}$) should be equal to 2^k. The error $E^{(5)}_{A_0,A_k,i,b}$ measures, for the tested noise estimator, the absolute deviation from the ideal value 2^k at each bin. To get a mean estimation of the error $E^{(5)}_{A_0,A_k,i,b}$ along all the test images and bins in figure 10, we define another error function as

$$E^{(6)}_{A_0,A_k,b} := \frac{1}{|I||B_k|}\sum_{i=1}^{|I|}\sum_{b=1}^{|B_k|} E^{(5)}_{A_0,A_k,i,b} \tag{16}$$

Table 5 shows the obtained mean down-scale error $E^{(6)}_{A_0,A_k,b}$ for the raw images in figure 10 depending on the scale k. The measurements are done for one of the green channels of the raw images.

$\boldsymbol{k}$	1	2	3
$E^{(6)}_{A_0,A_k}$	0.158	0.922	3.416

Table 5: Evaluation $E^{(6)}_{A_0,A_k,b}$ for the raw images in figure 10 depending on the scale k. The measurements are done for one of the green channels of the raw images.

5 Optimal Parameters

The optimal parameter choice depends on the size of the image. Three possible sizes were fixed: $S_0 = 6M$, $S_1 = \frac{S_0}{4} = \frac{6M}{4}$, $S_2 = \frac{S_1}{4} = \frac{6M}{16}$, $S_3 = \frac{S_2}{4} = \frac{6M}{32}$ and $S_4 = \frac{S_3}{4} = \frac{6M}{128}$, where M stands for megapixels. Table 6 shows the choice of the parameters values according to the size of the image.

Image size	Percentile p	Block size $w \times w$	Pre-filter operator $\mathbf{R}$
S_0	0.005	21×21	DCT supp. 7×7
S_1	0.005	15×15	DCT supp. 7×7
S_2	0.005	15×15	DCT supp. 7×7
S_3	0.005	15×15	DCT supp. 7×7
S_4	0.005	5×5	$\Delta\Delta\Delta$

Table 6: Best percentile p, block size $w \times w$, and pre-filter operator $\mathbf{R}$ for the Percentile method according to the size of the image.

6 Complexity Analysis of the Algorithms

Algorithm 1 creates a matrix of $(N_x - 2b \times N_y - 2b)$ elements and then fills matrix $\mathbf{V}$ with $N_y - b - 1 \times N_x - b - 1$ values. Therefore, the complexity is linear, $O(N)$ with $N \sim N_x \times N_y$. The noise estimation procedure (algorithm 2) computes the convolution $\tilde{\mathbf{U}} * \mathbf{F}$ in the Fourier domain using the FFT algorithm and then crops the result using algorithm 1, that has linear complexity. The complexity of computing the convolution is $O(N \log N)$, where $N \sim (N_x - s + 1) \times (N_y - s + 1)$. Therefore, the complexity of cropping the convolution is $O(N \log N)$. Then, algorithm 1 iterates M times through a loop that reads the blocks $\mathbf{W}^f_m$ and computes its variance. Since s is fixed, the complexity of the loop is linear with the number of iterations, $O(M)$, where $M = (N_x - w - s + 1) \times (N_y - w -$

$s+1$) is the number of overlapping blocks. After the execution of this loop, the SORTED function (implemented with the Quicksort algorithm) is called. Thus, the sorting operation has complexity $O(M \log M)$. Therefore, algorithm 2 has complexity $O(M \log M)$. Algorithm 3 first executes the *argsort* (implemented with the Quicksort algorithm) operation with complexity $O(N \log N)$. The loop that iterates through idx $= 0 \dots N$ just copies data in linear time $O(N)$. Therefore, algorithm 3 is executed with complexity $O(N)$. Algorithm 4 is simply a conditional comparison and then simple arithmetic operations. Therefore, algorithm 4 is executed in constant time $O(1)$. Algorithm 6 loops over the number of bins of the noise curve and inside the loop algorithm 4 is called. Since algorithm 4 is executed in constant time $O(1)$, algorithm 6 has a linear complexity $O(B)$, where B is the number of bins. Algorithm 7 loops over all possible pixels in the image (with the exception of the boundary of the image). The loop iterates through all the channels of the image and looks for groups of four connected pixels. Therefore, the inner loop is executed in linear time with the number of channels, $O(\text{num\_channels})$. Since the number of channels is small, it can actually be considered executed in constant time $O(1)$ once the number of channels has been fixed. The complexity of algorithm 7 is given by its main loop, that is executed in linear time with complexity $O(M)$.

7 Online Demo

An online demo is available for this algorithm in the IPOL web page of this article[3]. The users can upload any image to measure its noise. The demo also offers several types of pre-uploaded images to test the algorithm:

- Raw images obtained by splitting the raw channels R, G_1, G_2, B and leaving out the G_2 channel. Then, an RGB image is formed by using the R, G_1, B channels. Since in the raw image no gamma correction has been done yet, the values of the image are multiplied by 32 to increase their dynamics and screen visibility. The colors of these images are not quite adapted to human visualization, because no white balance has been applied to them.
- The JPEG versions of the same raw images, as they are encoded by the camera.
- Various JPEG images.
- High SNR raw images, down-scaled by eight with their color channels averaged, so that they are nearly noiseless. In the demo they are referred to as "no noise" images.

Once an image has been chosen, the following parameters can be configured:

Percentile. The possible values are 0.01%, 0.1%, 0.5% (default), 5%, 10% and 50%.

Pre-filter operators. Operator **R** whose stencil **F** is used to convolve the image with. The possible operators are: Identity (no filter), Directional derivative, Laplace, Laplace (2 iterations), Laplace (3 iterations), Laplace (4 iterations), DCT with support 7×7 (default), DCT with support 5×5, DCT with support 3×3 and the filter of the article Fast Noise Variance Estimation [5].

Block size. The size of the block. The possible choices are 3×3, 7×7, 8×8, 15×21 and 21×21 (default).

Curve filter iterations. It indicates the number of filtering iterations that are applied to filter the noise curve (see section 3.2). Default: five iterations.

[3] https://doi.org/10.5201/ipol.2013.90

Treatment of groups (2×2) of equal pixels. It allows to choose between ignoring the blocks that contain a group of four equal pixels in any channel (default), or using all the blocks unconditionally (see section 3.3).

Number of bins. It is the number of bins in the noise curve (see section 3.1). The number of bins that are used depends on the size of the image when "automatic selection" is chosen. First, a nearest compatible size of the image is considered (see section 5). For images whose size is S_0, S_1 or S_2, the number of bins is given by dividing the total number of pixels of the image by 42000. If the image is compatible with the size S_3, 4 bins are used. If the image is compatible with the size S_4 or if it is smaller, a single bin is used. Default: automatic selection.

A and B noise parameters. Add a simulated noise with variance $A + B\tilde{\mathbf{U}}$ to the input image. If $A = B = 0$ no noise will be added. If $B = 0$ homoscedastic noise with variance A will be added. Default: $A = B = 0$.

7.1 Subtraction of the Quantization Noise

In the online demo, all the images are encoded using 8 bits/pixel/channel. This adds a quantization error over the estimated noise that must be subtracted. Indeed, the variance of a uniform random variable in $[-1/2, 1/2]$ is

$$\sigma_q^2 = \int_{-\frac{1}{2}}^{\frac{1}{2}} (x - \bar{x})^2 \, \mathrm{d}x = \int_{-\frac{1}{2}}^{\frac{1}{2}} x^2 \, \mathrm{d}x = \left[\frac{x^3}{3}\right]_{-\frac{1}{2}}^{\frac{1}{2}} = \frac{1}{12}.$$

This is the variance of the quantization error that must be subtracted at each scale. The standard deviation of the noise is computed at each bin as the square root of the noise variance computed directly by the algorithm minus the variance of the variance of the (independent) quantization error. At each scale k the variance is divided by 4^k and thus the corrected standard deviation of the noise given by the demo is

$$\tilde{\sigma}_k = \sqrt{\hat{\sigma}_k^2 - \frac{\sigma_q^2}{4^k}} = \sqrt{\hat{\sigma}_k^2 - \frac{1}{4^k 12}}. \tag{17}$$

7.2 Example: *traffic* Image

The results of this example can be reproduced by adding noise with parameters $A = 0$ and $B = 0.5$ to the *traffic* image. The rest of the parameters are the default parameters of the demo. Figure 17 shows the input noiseless image *traffic* before adding signal dependent noise with variance $\sigma^2 = 0.5\mathbf{U}$. Figure 18 shows the noise estimated for the three first scales of the signal-dependent noise with variance $\sigma^2 = 0.5\mathbf{U}$ added to the *traffic* image. Because the noise was added to a noise-free image, we can compute the RMSE for the different scales S_0, S_1 and S_2, and the corresponding errors are 0.15, 0.18 and 0.16, respectively. Note that, as expected, the noise standard deviation is divided by approximately two when down-scaling the image by the same ratio.

Acknowledgements

Research partially financed by the MISS project of Centre National d'Etudes Spatiales, the Office of Naval Research under grant N00014-97-1-0839, by the European Research Council, advanced grant "Twelve labours" and the Spanish government under TIN2011-27539.

Figure 17: Noise free input image *traffic* before adding noise with variance $\sigma^2 = 0.5\mathbf{U}$.

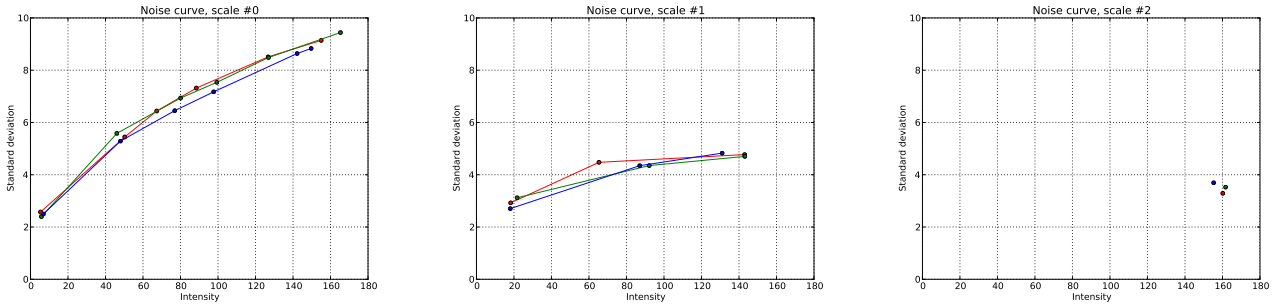


Figure 18: The noise estimated for the three first scales of the signal-dependent noise with variance $\sigma^2 = 0.5\mathbf{U}$ added to the *traffic* image. From left to right: scales S_0 (original), S_1 and S_2.

Image Credits


Miguel Colom, CC-BY


The USC-SIPI Image Database [13].

References

[1] A. Buades, Y. Lou, J.M. Morel, and Z. Tang. A note on multi-image denoising. In *International Workshop on Local and Non-Local Approximation in Image Processing*, pages 1–15. IEEE, 2009, http://dx.doi.org/10.1109/LNLA.2009.5278408.

[2] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, pages 297–301, 1965, http://dx.doi.org/10.2307/2003354.

[3] A. Foi, S. Alenius, and K. Katkovnik, V.and Egiazarian. Noise measurement for raw-data of digital imaging sensors by automatic segmentation of non-uniform targets. *IEEE Sensors Journal*, 7(10):1456–1461, 2007, http://dx.doi.org/10.1109/JSEN.2007.904864.

[4] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian. Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing*, 17(10):1737–1754, 2008, http://dx.doi.org/10.1109/TIP.2008.2001399.

[5] J. Immerkaer. Fast noise variance estimation. *Computer Vision and Image Understanding*, 64(2):300–302, 1996, http://dx.doi.org/10.1006/cviu.1996.0060.

[6] M. Lebrun, M. Colom, A. Buades, and J.M. Morel. Secrets of image denoising cuisine. *Acta Numerica*, 21:475–576, 2012, http://dx.doi.org/10.1017/S0962492912000062.

[7] J.S. Lee and K. Hoppel. Noise modelling and estimation of remotely-sensed images. *Proceedings of the International Geoscience and Remote Sensing Symposium*, 2:1005–1008, 1989, http://dx.doi.org/10.1109/IGARSS.1989.579061.

[8] G.A. Mastin. Adaptive filters for digital image noise smoothing: an evaluation. *Computer Vision, Graphics, and Image Processing*, 31(1):103–121, 1985, http://dx.doi.org/10.1016/S0734-189X(85)80078-5.

[9] P. Meer, J.M. Jolion, and A. Rosenfeld. A fast parallel algorithm for blind estimation of noise variance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):216–223, 1990, http://dx.doi.org/10.1109/34.44408.

[10] S.I. Olsen. Estimation of noise in images: an evaluation. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 55(4):319–323, July 1993. http://dx.doi.org/10.1006/cgip.1993.1022.

[11] N.N. Ponomarenko, V.V. Lukin, M.S. Zriakhov, A. Kaarna, and J.T. Astola. An automatic approach to lossy compression of AVIRIS images. In *International Geoscience and Remote Sensing Symposium*, pages 472–475. IEEE International, 2007, http://dx.doi.org/10.1109/IGARSS.2007.4422833.

[12] K. Rank, M. Lendl, and R. Unbehauen. Estimation of image noise variance. In *Vision, Image and Signal Processing, IEEE Proceedings*, volume 146, pages 80–84. IET, 1999, http://dx.doi.org/10.1049/ip-vis:19990238.

[13] Allan G Weber. The USC-SIPI image database version 5. *USC-SIPI Report*, 315:1–24, 1997. `http://sipi.usc.edu/database/`.