IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2022–10–18.
Submitted on 2022–05–10, accepted on 2022–10–03.
ISSN 2105–1232 © 2022 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.2022.405

Breaking down Polyblur: Fast Blind Correction of Small Anisotropic Blurs

Thomas Eboli, Jean-Michel Morel and Gabriele Facciolo

Université Paris-Saclay, ENS Paris-Saclay, CNRS, Centre Borelli, France
thomas.eboli@ens-paris-saclay.fr

Communicated by Julie Delon *Demo edited by* Thomas Eboli

Abstract

Polyblur is a two stage blind deblurring technique for removing small-sized blurs, like small camera shake or the lens point-spread function, proposed in 2021 by Delbracio et al. First, the blur is modeled with a zero-mean anisotropic Gaussian kernel whose parameters are rapidly estimated from the oriented blurry image gradients. Second, a sharp estimate is obtained by applying an approximate deconvolution filter, which is designed as a polynomial function of the estimated blurring kernel. Since in practice true blurs are not exactly Gaussian filters, the residual blur is gradually removed by repeating this two-stage procedure. Because it relies only on simple image manipulations, Polyblur is a quick blind deblurring technique, running in a fraction of a second on a smartphone. In this presentation, we analyze its key ingredients, showcase several use cases on real images, and provide Numpy and Pytorch implementations.

Source Code

The reviewed source code of this algorithm is available on the web page of this article[1]. The latest version of the code can be found at the Github repository[2].

Keywords: blind deblurring; spatial Gaussian filter; computational photography; sharpening; defocus; point-spread function

1 Introduction

Fast and efficient deblurring is pivotal in several image processing applications, for instance to correct camera shake [17], out-of-focus blur [19], or optical aberrations [14]. The restoration algorithms run most of the time in a blind setting where the underlying blur is unknown, and sometimes is even hard to model. Recent years have seen the advent of several blind deblurring techniques, which almost always embed two sub-modules: (i) A blur estimation algorithm applied to the degraded image at

[1] https://doi.org/10.5201/ipol.2022.405
[2] https://github.com/teboli/polyblur

hand, and (ii) a state-of-the-art non-blind deblurring method based on an elaborated image prior. In particular, the first one may leverage the properties of the blur kernel in the Fourier domain as proposed by Goldstein and Fattal [8, 1], or the so-called "unnatural" image priors [18, 12, 2] favoring latent sharp images with sharp edges. Both approaches are designed to predict any sort of blur trajectories, and are thus showcased for complex motion blurs. However, such approaches are generally slow, taking several seconds to minutes for even low-definition images, and are thus reserved to post-processing applications.

In this work we analyze Polyblur, a blind deblurring method proposed by Delbracio et al. [5] that addresses the correction of "mild" blurs: blurs whose kernels have relatively small supports and are responsible of slight defocus for instance, or accounting for the lens blur [10, 6]. The out-of-focus photograph in Figure 1 illustrates an instance of such "mild" blur. By focusing on such specific but common sort of blurs, Delbracio et al. introduce a fast and effective iterative deblurring technique that may even run on a mobile phone in a fraction of a second, narrowing the gap with real-time applications. This method consists in alternating between fast spatial Gaussian kernel estimation and non-blind deconvolution. The first stage leverages the possibility to approximate small blurs with Gaussian filters, whose covariance matrix can be directly estimated from the gradients of the blurry image. The latter exploits a truncated geometric series of the blur kernel approximating its theoretical inverse filter, whose convergence is guaranteed since Polyblur addresses only "mild" blurs removal. This truncated series yields a parametric deconvolution filter. Repeating these two steps a couple of times results in a sharper image.

(a) Defocused original photograph.

(b) Image restored with Polyblur.

Figure 1: Out-of-focus blur removal in a real-world deblurring scenario, featuring a "mild blur": We took a slightly defocused photograph of a peacock, resulting in small loss of sharpness of the bird. Polyblur, the method discussed in this presentation, is a fast iterative blind deblurring technique, i.e., we contemplate a realistic situation where a blur kernel is not available, to address the correction of such relatively small blurs. The resulting image is a sharper version of the original photograph, as if it were taken with accurate focusing on the bird's eye.

Formally, let u and v be sharp and blurry $H \times W$ images (we assume a grayscale version in this presentation but the model holds for color images as well), k the ground-truth blur kernel (for instance the lens point-spread function), and ε is additive noise. The image formation model reads

$$v = s(k * u + \varepsilon), \quad \text{with } \varepsilon \sim \mathcal{N}(0, \sigma_n^2), \tag{1}$$

where $*$ is the discrete convolution of two 2D arrays, and s models saturation clipping the pixel values within [0,1]. Polyblur aims at estimating from the gradients of v a Gaussian filter g that approximates the possibly non-parametric kernel k, and use it to deblur v. Repeated several times, this procedure yields a sharp estimate of u.

Algorithm 1: `Polyblur`

Data: Image v, blur coefficients (C, b), polynomial coefficients (α, β), Polyblur iterations N, prefiltering coefficients (σ_s, σ_r), prefiltering iterations M, saturation threshold t
Result: Sharp estimate $\widehat{u}$

$n \leftarrow 0$;
for $n < N$ **do**
  $g \leftarrow$ `GaussianBlurEstimation`(v, C, b, t);
  $v \leftarrow$ `Deconvolution`$(v, g, \alpha, \beta, \sigma_s, \sigma_r, M)$;
  $n \leftarrow n + 1$;
end
$\widehat{u} \leftarrow v$;

The main deblurring function `Polyblur` is shown in Algorithm 1, highlighting the two main steps of Polyblur. Section 2 motivates the principal assumptions behind Polyblur, with numerical and theoretical validations. The functions `GaussianBlurEstimation` and `Deconvolution` are detailed in Section 3. Section 4 presents qualitative results for defocus and lens blur correction, and for sharpening of images predicted with a super-resolution (SR) algorithm, for instance in [11].

2 Preliminary Results

We discuss in this section the two main assumptions behind Polyblur. Delbracio et al. [5] first posit that the blurs of interest are small enough (what they call "mild" in their article), so that a deconvolution filter formulated as a truncated power series of the blur kernel achieves satisfactory visual results. This suggests that no image prior, e.g., the total variation [13], is necessary for efficient restoration in this context. Second, effective deconvolution can be carried out by approximating the underlying blur kernel k with spatial Gaussian filters, whose parameters are inferred from the gradients of the blurry image.

2.1 Deconvolution Filters for "Mild" Blurs

The first main idea of Delbracio et al. is to target relatively small blurs compared to typical blur estimation, e.g., [12, 8]. The key assumption on these "mild" blurs can be formally expressed as

$$||\delta - k||_2 < 1, \tag{2}$$

where δ is the identity filter for the non-cyclic convolution $*$. This criterion implies that the kernel k does not importantly degrade the original image. Provided Equation (2) is valid, the inverse filter k^{-1} exists and may be written as the following geometric power series in $\delta - k$

$$k^{-1} = \sum_{i=0}^{\infty} (\delta - k)^i, \tag{3}$$

where the exponent i in this presentation denotes i consecutive convolutions with the same filter, with the convention that the exponent $i = 0$ results in the Dirac filter δ.

Proof. Calling $h = \delta - k$, we know that the inverse to $\delta - h$ exists and can be expressed as a power series of h whenever $\|h\|_2 < 1$ is verified. Since we assume that k is a "mild" blur, this condition is

fulfilled and the inverse filter to $\delta - h = k$ exists, and reads

$$k^{-1} = (\delta - h)^{-1} = \sum_{i=0}^{\infty} h^i = \sum_{i=0}^{\infty} (\delta - k)^i \stackrel{\text{def}}{=} p(\delta - k), \tag{4}$$

where p is the polynomial resulting from the summation of the filters on the left hand of the equation. The second equality transforming an inverse into a convergent series is a property of a Neumann series. □

In practice, this ideal inverse filter can be only approximated by a truncated polynomial of degree d

$$p_d(k) = \sum_{i=0}^{d} (\delta - k)^i. \tag{5}$$

As a result, we predict a sharp image $\widehat{u}$ by applying the deconvolution filter $p_d(k)$ to v

$$\widehat{u} = p_d(k) * v = p_d(k) * k * u + p_d(k) * \varepsilon. \tag{6}$$

The polynomial p_d should at the same time remove the blur k and prevent noise amplification. To that end, Delbracio et al. use a custom polynomial with coefficients $a_0, a_1, \ldots, a_d$, such that

$$p_d(k) = \sum_{i=0}^{d} a_i k^i. \tag{7}$$

For a selected truncation degree d, the coefficients are chosen to reach the best deblurring/noise compensation trade-off. In Section 3, we detail the strategy for finding the polynomial coefficients when $d = 3$.

2.2 Quick Estimation of a Gaussian Blur

The second main assumption in [5] is that zero-mean spatial Gaussian filters can model well "mild" blurs to achieve efficient deblurring. Zero-mean Gaussian kernels are completely determined by their covariance matrix, which is itself shaped by three parameters: the standard deviations along the principal and orthogonal axes σ and ρ, and the principal axis orientation θ. These kernels read

$$g(x) = [\det(2\pi\Sigma)]^{-\frac{1}{2}} \exp\left(-\frac{1}{2} x^\top \Sigma^{-1} x\right), \tag{8}$$

for all x in $\mathbb{R}^2$, with covariance matrix

$$\Sigma = R(\theta)^\top \begin{bmatrix} \sigma^2 & 0 \\ 0 & \rho^2 \end{bmatrix} R(\theta), \tag{9}$$

where $R(\theta)$ denotes the 2D rotation matrix of angle θ. The triplet (σ, ρ, θ) thus fully determines the 2D Gaussian filter. We now show that these three parameters can be computed solely from the gradients of the blurry image v at hand.

Hypothesis on the orientation θ. First, let us defined the 2-infinite norm for an image u as

$$\|u\|_{2,\infty} \stackrel{\text{def}}{=} \max_{x \in D} \|u(x)\|_2, \tag{10}$$

where D is the image domain. Delbracio et al. observe that in sharp natural images, the infinite norm of the gradients is roughly the same in any direction, or formally

$$\|\nabla_{\varphi_i} u\|_{2,\infty} \approx \|\nabla_{\varphi_j} u\|_{2,\infty}, \tag{11}$$

for all orientations φ_i and φ_j in $[0, \pi)$. Blurring the image in a direction θ thus degrades the infinite norm of the oriented gradients in this orientation. The angle θ may therefore be obtained from the blurry image v with

$$\theta = \operatorname*{argmin}_{\varphi} \|\nabla_{\varphi} v\|_{2,\infty}. \tag{12}$$

Figure 2 shows the infinite norms of the directional gradients for a sharp image u and for its blurry version v, which was blurred with a Gaussian of parameters $\theta = \pi/2$, $\sigma = 3$ and $\rho = 0.5$, i.e., a vertically elongated filter, and contaminated with 1% additive Gaussian noise. The curves are the median of the infinite norms over the 500 images of the BSD dataset [3]. For the sharp image u, the infinite norm has a constant value of about 0.8 across all directions φ, whereas for its blurry counterparts a clear minimum is achieved when φ is equal to θ.

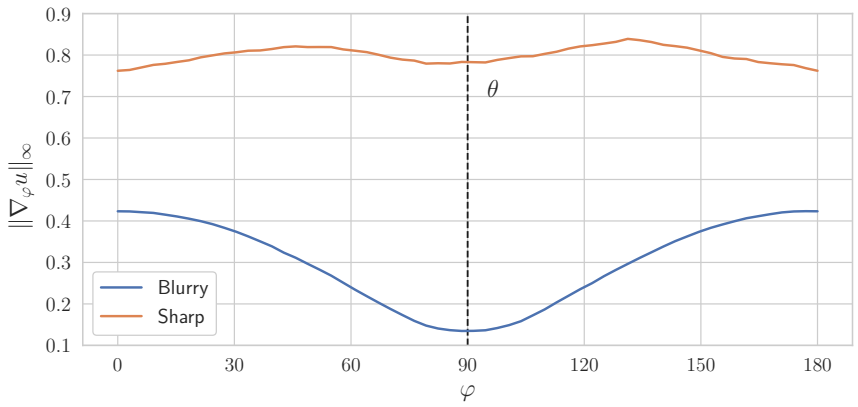


Figure 2: The median over 500 images from [3] of the infinite norm of the gradients of the sharp images and synthetic blurry counterparts convolved with a vertically elongated Gaussian filter, i.e., $\theta = \pi/2$. The magnitude of the gradient is roughly constant in all the directions for the sharp images, whereas it exhibits a minimum at θ for the blurry ones.

Hypothesis on the standard deviation values σ and ρ. Delbracio et al. [5] base their blur estimation procedure on the affine relationship between the variance of a Gaussian kernel and the squared magnitude of the gradients of a blurry edge. Here, we improve on the original paper and prove, partly with a numerical argument, such a claim for an ideal edge.

Proof. Let f be an ideal edge defined as $f(x) = 1(x > 0)$ for all x in $\mathbb{R}$, and g a 1D zero-mean Gaussian filter of variance σ^2. The zero-mean hypothesis means that the blur does not move the location of the maximum slope of the edge. Taking the gradient of the blurry edge $g * f$ yields

$$\nabla(g * f) = g * \nabla f = g * \delta = g, \tag{13}$$

where we have permuted the gradient and the blur operators, and used the property that the gradient of this ideal edge is the Dirac mass, approximated by the δ kernel defined above. Since the Gaussian

filter g is zero-mean, its maximum is reached for $x = 0$. The infinite norm of the gradient of a blurry ideal edge therefore is

$$\|\nabla(g * f)\|_\infty = \max_{x \in D} \|\nabla(g * f)(x)\| = g(0) = \frac{1}{\sqrt{2\pi}\sigma}. \tag{14}$$

In the noise-free case, there exists a *linear* relationship between the magnitude of the blur and the inverse of the gradient's value at the location of the steepest point of an edge.

In practice, the edge of an image is noisy, and we thus have to consider an additional noise vector. The actual gradient we consider is

$$\nabla(g * f + \varepsilon) = \nabla(g * f) + \nabla\varepsilon = g * \nabla f + \nabla\varepsilon. \tag{15}$$

We now compute the infinite norm of this quantity to obtain an analytic relationship between σ^2 and $1/\|\nabla(g * f + \varepsilon)\|_{2,\infty}$. Since we now include the noise in our analysis, we should actually take the expectation of the infinite norm for several instances of ε, which reads

$$\mathbb{E}\left[\|\nabla(g * f + \varepsilon)\|_{2,\infty}\right] = \mathbb{E}\left[\max_{x \in D} \|\nabla(g * f + \varepsilon)(x)\|^2\right], \tag{16a}$$

$$= \mathbb{E}\left[\max_{x \in D} \left((g * \nabla f)^2(x) + (\nabla\varepsilon)^2(x) + 2\nabla\varepsilon(x)(g * \nabla f)(x)\right)\right]. \tag{16b}$$

In general, one cannot permute the expectation and the max in the above equations. However, recall that we address "mild" blur removal, and on images with noise supposed relatively small. This is a reasonable assumption since deblurring/sharpening usually occurs after denoising in a camera pipeline, or on post-processed images. Figure 3(a) demonstrates the empirical commutation of the expectation and max applied to squared gradients of noisy and blurry images, under the mild noise assumption. We computed the expectation by averaging 100 instances of ε for each one of the 600 synthetic images used in this experiment. As a result, the expectation $\mathbb{E}$ and max are shown to commute in this "mild" blur and reasonable noise regime. The function n in the plot's labels is a normalization function featured in [5] and detailed in the next section in Equation (20). Inverting the application of expectation and max thus yields

$$\mathbb{E}\left[\|\nabla(g * f + \varepsilon)\|_{2,\infty}\right] = \max_{x \in D} \left((g * \nabla f)^2(x) + \mathbb{E}\left[(\nabla\varepsilon)^2(x) + 2\nabla\varepsilon(x)(g * \nabla f)(x)\right]\right) \tag{17a}$$

$$= \|g * \nabla f\|_{2,\infty} + B\sigma_n^2, \tag{17b}$$

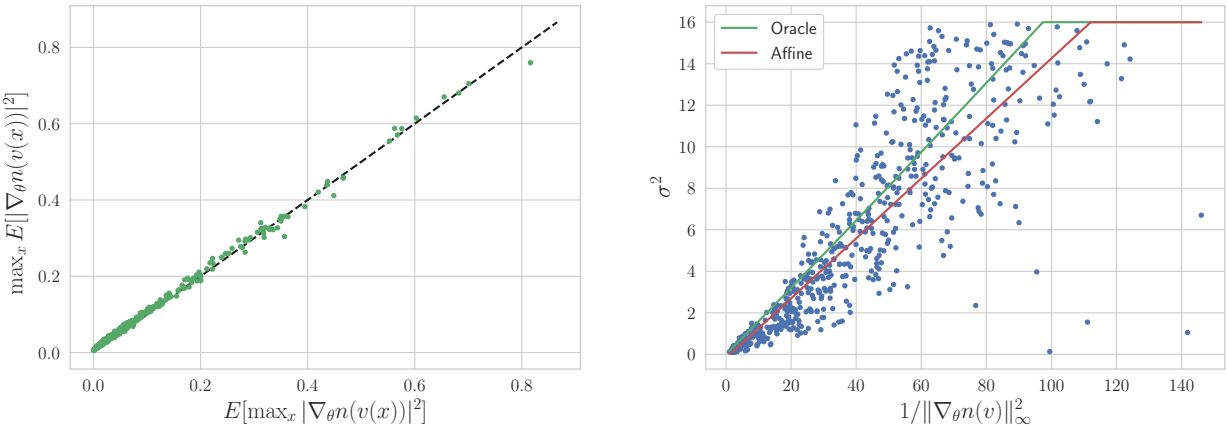
where we used the fact that $\mathbb{E}[\varepsilon] = 0$ implies that $\mathbb{E}[\nabla\varepsilon] = 0$, and where B is a constant depending on the implementation of ∇. Injecting the relationship of Equation (14) (with the relationship $\|u\|_{2,\infty} = \|u\|_\infty^2$), and isolating σ^2 results in

$$\sigma^2 = \frac{1}{2\pi\left(\mathbb{E}\left[\|\nabla(g * f + \varepsilon)\|_{2,\infty}\right] - B\sigma_n^2\right)}. \tag{18}$$

By writing $h^2 = 1/\mathbb{E}\left[\|\nabla(g * f + \varepsilon)\|_{2,\infty}\right]$ for conciseness, and by multiplying the numerator and the denominator by h, the previous relationship now reads

$$\sigma^2 = \frac{h^2}{2\pi(1 - B\sigma_n^2 h^2)}. \tag{19}$$

Figure 3(b) compares this "oracle" model (the green curve) with the affine model predicted with the protocol of Delbracio et al. (and detailed in the next section and Algorithm 3) on 600 synthetic blurry images. The oracle model has a linear behavior in this range of values, validating the affine model of [5] whose plot mimics that of the green curve. □



(a) Permuting max and $\mathbb{E}$.

(b) Oracle and affine models.

Figure 3: (a) From 600 synthetic images blurred with Gaussian filters whose standard deviation in the principal direction σ is smaller than 4, we show that computing first the expectation with respect to the noise followed by taking the max is equivalent to taking the max followed by the expectation of the square image gradients (the black dashed line is the identity function $x \mapsto x$). This suggests that max and the expectation can permute in the context of this presentation. The normalizing function n is defined later in Equation (20). (b) We follow the calibration protocol of [5] and use 600 synthetic images to fit an affine model (the red curve) between the reciprocal of the square infinite norm of the gradients of the image and the variance of the blur in the principal direction σ^2. We also compute the oracle model of Equation (19) (the green curve). The similarity between the two plots suggests that in low noise level and "mild" blur regime, the affine relationship of [5] is valid.

This proof is also valid for contrasted edges that are not Heaviside functions, solely changing the relationship of Equation (14) with the multiplication of a scalar that corresponds to the magnitude of the gradients at 0.

A key result of this proof is that the quality of the prediction highly depends on the noise intensity: If the noise is too important, the maximum value will change and lead to erroneous results. Furthermore, in practice the values of the slope and the intercept of this affine model will depend on the noise level, preventing the existence of noise-agnostic parameters. This algorithm is thus very sensitive to noise as stressed in Section 4.

3 Method

3.1 Gaussian Filter Estimation

The implementation of the function `GaussianBlurEstimation` is detailed in Algorithm 2. It consists in first estimating the Gaussian blur's principal direction θ based on the model in Equation (12) relating θ and the minimizer of the infinite norm of the gradients of the blurry image across the different directions. Second, this function leverages an affine relationship between the variance of the Gaussian blur and the norm of the gradients in the principal directions θ and $\theta + \pi/2$, illustrated by the red plot in Figure 3. This yields the variance values σ^2 and ρ^2.

Image normalization. The analysis of the blur estimation algorithm in the previous section supposes that the latent sharp edges are ideal Heaviside functions from 0 to 1, which is untrue in practice. Indeed, the intensity may vary across images. To compensate for this disparity, Delbracio

Algorithm 2: `GaussianBlurEstimation`

Data: Blurry image v, blur coefficients (C, b), saturation threshold t
Result: Gaussian filter g

`/* Compute the saturation mask */`
$m_t \leftarrow v < t$ `// Binary mask setting to 0 the pixels greater than` t
`/* Predict the blur's main direction */`
$\Phi_0 \leftarrow \{0, 30, 60, 90, 120, 150\}^\circ$;
$\nabla_\varphi n(v) \leftarrow \cos(\varphi)\nabla_x n(v) - \sin(\varphi)\nabla_y n(v), \quad \forall \varphi \in \Phi_0$;
`/*` $n(v)$ `is the normalized image defined in (20) */`
$\Phi_1 \leftarrow \{0, 6, 12, \ldots, 168, 174\}^\circ$;
$\widehat{\nabla_\varphi n(v)} \leftarrow$ `CubicInterpolation`$(\{\nabla_\phi n(v)\}_{\phi\in\Phi_0}, \varphi), \quad \forall \varphi \in \Phi_1$;
$\theta \leftarrow \operatorname{argmin}_{\varphi\in\Phi_1} \|\widehat{\nabla_\varphi n(v)} \odot m_t\|_\infty$;
`/* Estimate the blur's standard deviation values in main directions */`
$\sigma^2 \leftarrow C^2 \,/\, \|\nabla_\theta n(v) \odot m_t\|_\infty^2 - b^2$;
$\rho^2 \leftarrow C^2 \,/\, \|\nabla_{\theta+\frac{\pi}{2}} n(v) \odot m_t\|_\infty^2 - b^2$;
`/* Clip the predicted standard deviations to the range` $[0.3, 4]$ `*/`
$\sigma^2 \leftarrow$ `Clip`$(\sigma^2, 0.09, 16)$;
$\rho^2 \leftarrow$ `Clip`$(\rho^2, 0.09, 16)$;
`/* Build the spatial Gaussian filter */`
$\Sigma \leftarrow R(\theta)^\top \begin{bmatrix} \sigma^2 & 0 \\ 0 & \rho^2 \end{bmatrix} R(\theta)$;
$g(x, y) \leftarrow \exp\left(-\frac{1}{2}[x, y]\Sigma^{-1}[x, y]^\top\right) / \left(2\pi \det(\Sigma)\right), \quad \forall (x, y)$ in predefined support;

et al. propose to correct the contrast by normalizing the blurry image v with

$$n(v) = \min\left(\max\left(\frac{v - v[q]}{v[1-q] - v[q]}, 0\right), 1\right), \tag{20}$$

where $v[q]$ is the q-th quantile of the pixel values in the image v. In practice $q = 0.0001$ was found to yield satisfactory results.

Gradient computation. There exist many techniques to compute the image gradient, e.g., with finite differences. We favor a fast implementation leveraging the properties of the image derivatives in the Fourier domain, which may be seen as interpolating the gradients on a sine basis. By denoting U the 2D Fourier transform of u with spatial frequencies ξ and ν, we compute the horizontal and vertical gradients with the inverse 2D Fourier transform. Formally, for all (x, y) in $[0, W-1] \times [0, H-1]$

$$\nabla_x u(x, y) = \sum_{\xi=0}^{W-1} \sum_{\nu=0}^{H-1} j2\pi\xi U(\xi, \nu) \exp\left(j2\pi[\xi x + \nu y]\right), \tag{21a}$$

$$\nabla_y u(x, y) = \sum_{\xi=0}^{W-1} \sum_{\nu=0}^{H-1} j2\pi\nu U(\xi, \nu) \exp\left(j2\pi[\xi x + \nu y]\right). \tag{21b}$$

The derivative along a direction φ in $[0, 2\pi)$ is then computed as

$$\nabla_\varphi u(x, y) = \cos(\varphi)\nabla_x u(x, y) - \sin(\varphi)\nabla_y u(x, y). \tag{22}$$

Computing the gradients with this technique calls the fast Fourier transforms algorithm only three times: Once for computing $\widehat{u}$, and twice for computing the gradients $\nabla_x u$ and $\nabla_y u$. Computing

the image gradients with this approach may be slower than other typical techniques such as finite-difference filters or Sobel filters. However, amongst all the possible implementations, we have observed that the Fourier transform-based version is the best one in practice to exhibit the affine relationship between the norm of the image gradients and the standard deviation of the 1D slices of the blur kernel. It is thus our best option to replicate the results of the original paper [5].

Handling saturation. We modify in this presentation the original blur estimation routine of Delbracio et al. to circumvent the impact of the saturation operator s. Indeed, s breaks the typical linear convolution model accounting for the blur, and thus the underlying affine relationship pivotal to estimate the standard deviation of the blur. We take inspiration of deblurring techniques for saturated images introduced in e.g., [16], and introduce a binary mask m_t marking the pixel values above the threshold t in v, for instance 0.95 for pixel values between 0 and 1. The image gradients at the flagged pixel locations are set to 0, and are thus not considered in the infinite norm computation.

Direction θ estimation. We cannot compute the directional gradients in all the possible directions. We instead compute a coarse estimate with directions in $\Phi_0 = \{0, 30, 60, 90, 120, 150\}^\circ$. We interpolate with the `CubicInterpolation` algorithm the infinite norm values $\|\widehat{\nabla_\varphi n(v)}\|_\infty$ in new directions sampled every 6°, thus belonging to the set $\Phi_1 = \{0, 6, \dots, 174, 180\}^\circ$, and finally estimate the blur direction in Φ_1 as

$$\theta = \operatorname*{argmin}_{\varphi \in \Phi_1} \|\widehat{\nabla_\varphi n(v)}\|_\infty. \tag{23}$$

In our implementation, we use the `scipy.interpolate.interp1d` routine to compute the cubic interpolation. Having computed the principal direction of the blur, we estimate the standard deviation values σ and ρ, in the directions θ and $\theta + \pi/2$.

Estimation of the standard deviations σ and ρ. We leverage the affine model in Figure 3 to compute the standard deviation of the blur in the directions θ and $\theta + \pi/2$. Delbracio et al. learn the following affine model from supervisory data

$$\sigma^2 = \frac{C^2}{\|\nabla_\theta n(v)\|_\infty^2} - b^2 \quad \text{and} \quad \rho^2 = \frac{C^2}{\|\nabla_{\theta+\frac{\pi}{2}} n(v)\|_\infty^2} - b^2, \tag{24}$$

where C^2 and b^2 are the two learnable parameters accounting for the slope and the intercept of the affine relationship. In particular, b accounts for the noise level as detailed in the previous section, but also includes the interpolation errors of `CubicInterpolation` in Algorithm 2, as discussed in [5]. We calibrate the model with M synthetic images blurred with Gaussian filters oriented in randomly sampled directions in $[0, \pi)$, resulting in training pairs (σ_i, v_i) $(i = 1, \dots, M)$. The corresponding optimization problem is

$$\min_{C^2, b^2} \sum_{i=1}^{M} \left\| \sigma_i^2 - \frac{C^2}{\|\nabla_\theta n(v_i)\|_\infty^2} + b^2 \right\|_1. \tag{25}$$

Minimization was carried out with the simplex algorithm of `scipy.optimize.linprog`, solving a linear programming reformulation of (25) [4, Sec 1.2.2]. Algorithm 3 details our implementation.

Figure 4 shows our predictions for C and b by solving Equation (25). We used 60 sharp images from the DIK2K validation dataset, discarding 40 images over the 100 composing this set that feature motion or defocus blur. Following Delbracio et al. [5] we blurred these sharp images with Gaussian filters, where θ was randomly sampled in $[0, \pi)$, and σ and ρ randomly sampled in $[0.3, 4]$. We lastly added Gaussian noise with $\sigma_n = 0.01$, resulting in blurry and noisy images. We sampled 10 kernels per image, which yielded a set of $M = 600$ blurry images with corresponding known variances σ^2 and

Algorithm 3: `CalibratingAffineModel`

Data: Infinite norms $\|\nabla_\theta n(v_1)\|_\infty, \dots, \|\nabla_\theta n(v_M)\|_\infty$, corresponding to blur variances $\sigma_1, \dots, \sigma_M$

Result: Optimal coefficients (C^2, b^2)

`/* Setup the linear cost` $c^\top \mathbf{z}$ `*/`

$c \leftarrow$ `zeros`$(M+2, 1)$ `//` $(M+2) \times 1$

$c[0:M] \leftarrow 1$;

`/* Setup the linear inequality constraint` $A\mathbf{z} \leq b$ `*/`

$$\mathbf{x} \leftarrow \begin{bmatrix} \|\nabla_\theta n(v_1)\|_\infty^{-2} & \|\nabla_\theta n(v_2)\|_\infty^{-2} & \dots & \|\nabla_\theta n(v_M)\|_\infty^{-2} \\ 1 & 1 & \dots & 1 \end{bmatrix}^\top \text{ // } M \times 2$$

$I \leftarrow$ `eye`(M) `//` $M \times M$

$$A \leftarrow \begin{bmatrix} -I & \mathbf{x} \\ -I & -\mathbf{x} \end{bmatrix} \text{ // } 2M \times (M+2)$$

$$\mathbf{y} \leftarrow \begin{bmatrix} \sigma_1^2 & \dots & \sigma_M^2 \end{bmatrix}^\top \text{ // } M \times 1$$

$$b \leftarrow \begin{bmatrix} \mathbf{y} & -\mathbf{y} \end{bmatrix}^\top \text{ // } 2M \times 1$$

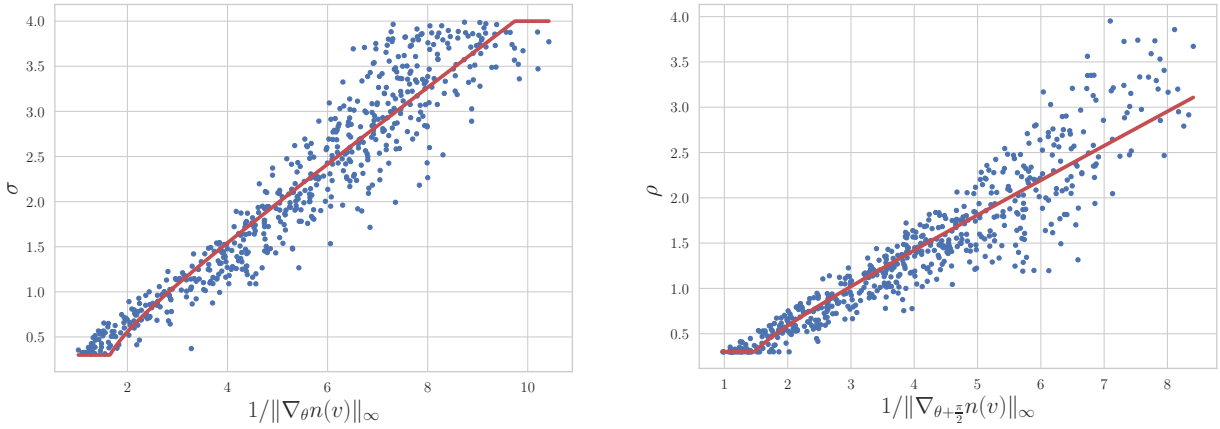
`/* Do optimization of` $\min_{\mathbf{z}} c^\top \mathbf{z}$ `s.t.` $A\mathbf{z} \leq b$ `and` $\mathbf{z} \geq 0$ `*/`

$\mathbf{z} \leftarrow$ `linprog`(c, A, b) `//` $(M+2) \times 1$

$C^2 \leftarrow \mathbf{z}[-2]$;

$b^2 \leftarrow \mathbf{z}[-1]$;

ρ^2 and directions θ and $\theta + \pi/2$. We show in both plots that one can fit a red curve corresponding to the square root of the affine function parameterized by the optimal C^2 and b^2 values, which are found by solving Equation (25). With this setting, we found $C = 0.362$ and $b = 0.468$, which are close to the values claimed by Delbracio et al. that are $C = 0.352$ (when reported to the [0,1] pixel value range) and $b = 0.768$.



(a) Normal direction θ.

(b) Orthogonal direction $\theta + \frac{\pi}{2}$.

Figure 4: Calibration curves obtained with `CalibratingAffineModel` (in Algorithm 3). We used $M = 600$ synthetic blurry images (each blue point), convolved with Gaussian filters whose variances in the principal directions σ^2 and ρ^2 are known. We added Gaussian noise with $\sigma_n = 0.01$ to these samples. The red curve is the squared root of the affine function parameterized by C^2 and b^2, fitted from our synthetic samples in the directions θ and $\theta + \pi/2$.

In practice the range of acceptable values for σ and ρ is kept within $[0.3, 4]$, which covers most of the magnitudes of mild blurs according to the original paper [5]. Beyond this range, the gradient magnitudes are deemed either too large (most likely for sharp images) or too small (most likely for

extremely blurry images, which goes beyond the scope of mild blurs). In these cases, we follow Delbracio et al. and simply clip the predicted standard deviation values to the range $[0.3, 4]$.

The three quantities θ, σ and ρ are then used to build the covariance matrix (9), and thus the Gaussian blur g accounting for the true kernel k in the formation model (1).

3.2 Mild Blur Removal

As seen in Section 2.1 and the criterion (2), the "mild" blurs are small corruptions for which no image prior, e.g., the total variation [13], is necessary to predict a natural image. As a result, a simple method based on a deconvolution filter may be enough to recover sharp details.

Delbracio et al. use the following polynomial to approximate the inverse of the small Gaussian filter g

$$p_{3,\alpha,\beta}(g) = \sum_{i=0}^{3} a_{i,\alpha,\beta} g^i = (\alpha/2 - \beta + 2)g^3 + (3\beta - \alpha - 6)g^2 + (5 - 3\beta + \alpha/2)g + \beta\delta, \tag{26}$$

where g^i corresponds to convolving i times g with itself. This parameterization comes from constraints on the values a polynomial filter must satisfy to prevent unwanted transformations such as luminance changes or a too important noise amplification. These constraints are derived in the original paper [5].

Figure 5 shows the different behaviors of $p_{3,\alpha,\beta}$ depending on α and β. The parameters α and β control how much blur is removed in the mid and high-frequencies. In particular β may amplify remaining noise when boosting the higher frequencies. When $\alpha = 2$ and $\beta = 4$, $p_{3,2,4}$ boils down to the truncated Neumann series in Equation (5) with degree $d = 3$. In practice, Delbracio et al. explain that α and β can have any value close to this setting. In this paper, we use these values for images with very small noise or compression artifacts, and switch to $\alpha = 6$ and $\beta = 1$ when noise and artifacts are more important: We favor boosting the mid-frequencies to improve sharpness but not as much the high-frequencies to keep low the noise and artifacts, as done in [5].

Note that a polynomial filter with a greater truncation degree, e.g., $d = 10$ in Figure 5, can boost the mid and high-frequencies at the same time, without overshooting over 1 when computing $p_d(g) * g$, whereas setting α and β in Equation (26) amounts to a trade-off between overshooting in the mid and/or the high-frequencies for the given truncation degree $d = 3$. The deblurred image z is obtained with

$$z = p_{3,\alpha,\beta}(g) * v. \tag{27}$$

Instead of explicitly constructing the polynomial $p_{3,\alpha,\beta}$, we use the efficient polynomial filtering implementation of [5], detailed in Algorithm 4, which computes just 3 convolutions with g.

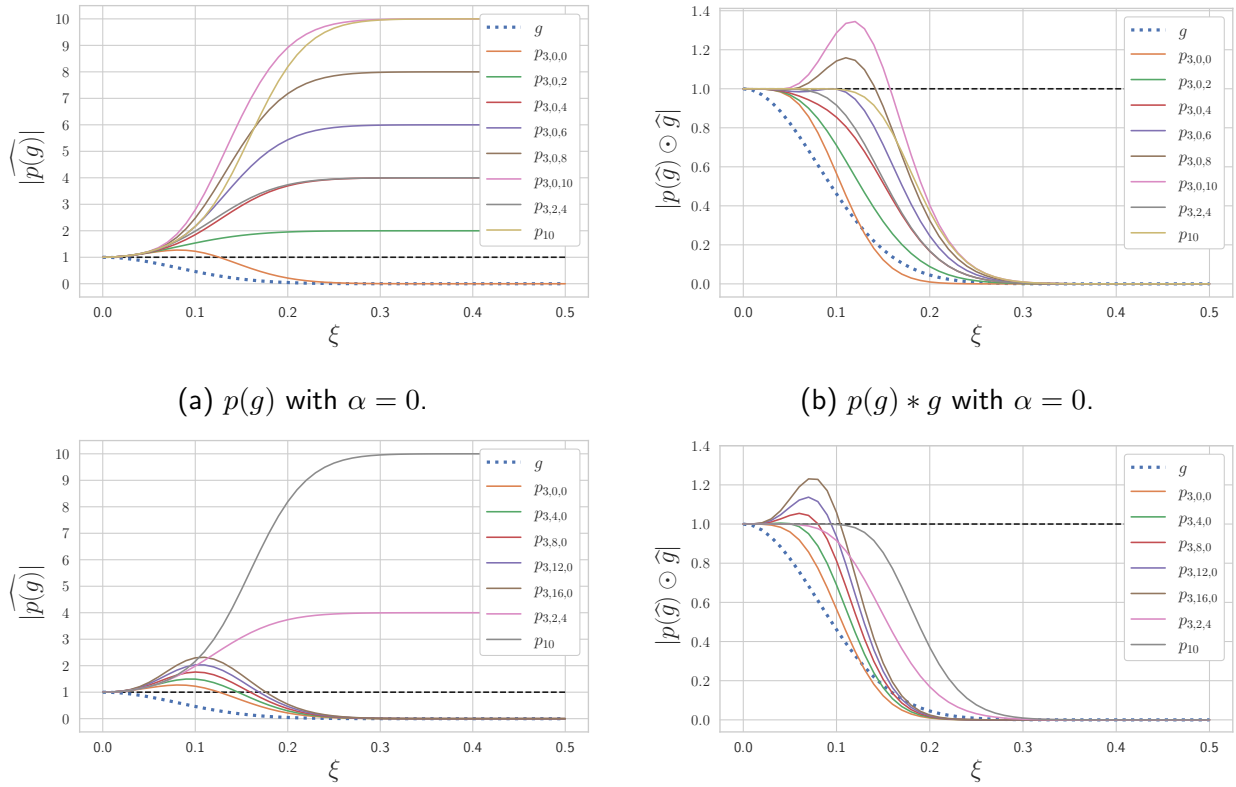
Halo removal. The deblurring filter may introduce a few halos next to the salient edges, caused by the inversion of the gradients between the blurry image and its filtered counterpart, or formally

$$-\nabla v(x, y)\nabla z(x, y) > 0, \tag{28}$$

for certain pixel locations (x, y) in $[0, W-1] \times [0, H-1]$. This may be compensated with a masking technique proposed in [5], whose pseudo-code `HaloRemoval` is detailed in Algorithm 5. It consists in a linear combination of the halo-free blurry image v and the filtered version z, whose $H \times W$ weights γ are computed based on the criterion in Equation (28) (see line 2 of Algorithm 5)

$$\widehat{u} = (1 - \gamma) \odot v + \gamma \odot z, \tag{29}$$

where $\odot$ is the pixelwise multiplication.



(a) $p(g)$ with $\alpha = 0$.

(b) $p(g) * g$ with $\alpha = 0$.

(c) $p(g)$ with $\beta = 0$.

(d) $p(g) * g$ with $\beta = 0$.

Figure 5: Profiles of several versions of the discrete Fourier transforms of $p_{3,\alpha,\beta}(g)$ and $p_{3,\alpha,\beta}(g) * g$ when α and β vary. The graphs show that α boosts the mid-frequencies, causing overshooting for values above 4. They also show that β boosts the high-frequencies, thus also magnifying the possibly remaining noise. The couple $(\alpha, \beta) = (2, 4)$ exactly leads to the truncation of the polynomial (3) to the degree 3, i.e., $p_{3,2,4} = p_3$ as defined in Equation (3) with $d = 3$. We have also plotted the truncation to degree 10, closer to the exact inverse filter, and boosting at the same time the mid and high frequencies without any overshoot. We also plot the profile of $\widehat{g}$ (the dotted blue curve), showing it decays the mid-frequencies and set to 0 the highest frequencies above 0.2.

Iterative deblurring. Unlike a typical FFT-based inverse filter, $p_{3,\alpha,\beta}$ is a rather conservative filter preventing the typical ringing artifacts, but leaving some blur in the restored image. The two-stage technique of [5] and detailed in this presentation may be applied several times by re-estimating the remaining blur with the Gaussian blur estimation technique in Section 3.1, and removing it with the polynomial filter $p_{3,\alpha,\beta}$. This is represented in Algorithm 1 by the for loop repeating the two main routines `GaussianBlurEstimation` and `Deconvolution`. Figure 6 shows an example of iterative deblurring. After one iteration, contrast is enhanced but residual blur is noticeable. After a second iteration, most of this residual is removed. A third iteration does not further improve sharpness and magnifies instead the noise, thus degrading the image quality.

Prefiltering to handle noise and compression artifacts. As shown in Figure 6 and as explained in the previous paragraph, Polyblur amplifies the noise and/or compression artifacts since it does not rely on any image prior. To prevent amplifying noise Delbracio et al. propose to first split the blurry image into base and texture components with the edge-aware recursive filter of [7]. This is a lightweight alternative to denoising the image with a convolutional neural network (CNN), at the cost of some smoothing and some additional memory requirement. The texture image contains the noise and the quantification artifacts, whereas its base counterpart is a noise-free smoothed version with sharp edges. The restored image is thus obtained by running Polyblur on the base image, and

Algorithm 4: `Deconvolution`

Data: Blurry image v, Gaussian kernel g, polynomial parameters (α, β), prefiltering parameters (σ_s, σ_r), prefiltering iterations M
Result: Deblurred estimate u

`/* Split the blurry image into noise residual and smoothed base image */`
$w \leftarrow$ `EdgeAwareSmoothing`$(v, \sigma_s, \sigma_r, M)$;
$v_n \leftarrow v - w$;
`/* Apply the polynomial deconvolution filter */`
`/* The coefficients` $a_{i,\alpha,\beta}$ `are the ones given in Equation (26) */`
$z \leftarrow a_{3,\alpha,\beta} w$;
for $i \in \{0, 1, 2\}$ **do**
  $z \leftarrow g * z + a_{2-i,\alpha,\beta} w$;
end
`/* Correct the ringing artifacts caused by misprediction of the blur kernel */`
$u \leftarrow$ `HaloRemoval`(z, w);
`/* Add back the noise residual */`
$u \leftarrow u + v_n$;

Algorithm 5: `HaloRemoval`

Data: Deblurred image z, blurry image v
Result: Halo-free estimate u

$m \leftarrow \langle -\nabla v, \nabla z \rangle$;
$\gamma \leftarrow \max\left(m / (\|\nabla v\|_2^2 + m), 0\right)$;
$u \leftarrow (1 - \gamma) \odot v + \gamma \odot z$;

(a) Blurry. (b) 1 iteration. (c) 2 iterations. (d) 3 iterations.

Figure 6: Qualitative comparison of the deblurred results for several iterations of Polyblur (N in Algorithm 1) applied to a synthetic blurry image with Gaussian noise with $\sigma_n = 0.01$. The sharpest result is obtained after two iterations. A third one marginally enhances sharpness, but magnifies the noise.

adding back the textures. In our implementation of [7] detailed in Algorithms 6 and 7, we used the parameters $\sigma_r = 0.8$, $\sigma_s = 2$, and $M = 3$. Figure 7 shows deblurring examples after three iterations of Polyblur on a synthetic image with additive noise and JPEG compression artifacts. The version without prefiltering exhibits amplified noise (for instance in the background),whereas the variant with prefiltering has barely visible artifacts. Note that in the original paper [7] three implementations of the domain transform algorithm are proposed, including the recursive version detailed in this article. Bilateral filtering is another possible cheap candidate for this base-detail decomposition.

Handling spatially-varying blurs. In practice, the blur may vary across the image, for instance due to the lens point-spread function [14] or camera shake [17]. We follow Hirsch et al. [9], and approximate a non-uniform blur by locally-uniform ones. We thus run the vanilla Polyblur algorithm (in Algorithm 1) on overlapping patches of v that we call v_p, with p in P the set of cropping indices.

(a) Blurry (28.31dB). (b) W/o prefiltering (28.28dB). (c) With prefiltering (28.82dB).

Figure 7: Qualitative comparison of the deblurred results after 3 iterations, but with and without the prefiltering technique based on [7]. The numbers in parenthesis are the PSNR scores computed with the original sharp image. The image is synthetically blurred, and further corrupted with Gaussian noise with $\sigma_n = 0.01$ and JPEG compression set to 85%. The variant of Polyblur without prefiltering magnifies the combination of noise and compression artifacts whereas with it, the image is deblurred and artifacts are kept with small magnitudes. The PSNR scores also indicate prefltering helps to increase visual quality in presence of noise/compression artifacts.

Algorithm 6: `EdgeAwareSmoothing`

Data: Image I, weights (σ_s, σ_r), filter iterations M
Result: Smoothed image F

$h, w, c \leftarrow$ `shape`(I);
`/* Compute the horizontal and vertical domain transform derivatives */`
$I_x \leftarrow$ `zeros`(h, w);
$I_y \leftarrow$ `zeros`(h, w);
$j \leftarrow 1$;
for $j < w$ **do**
  $I_x(:, j) = |I(:, j, 0) - I(:, j-1, 0)| + |I(:, j, 1) - (:, j-1, 1)| + |I(:, j, 2) - I(:, j-1, 2)|$;
  $j \leftarrow j + 1$
end
$i \leftarrow 1$;
for $i < h$ **do**
  $I_y(i, :) = |I(i, :, 0) - I(i-1, :, 0)| + |I(i, :, 1) - I(i-1, :, 1)| + |I(i, :, 2) - I(i-1, :, 2)|$;
  $i \leftarrow i + 1$
end
$H \leftarrow (1 + \sigma_s/\sigma_r \times I_x)$;
$V \leftarrow (1 + \sigma_s/\sigma_r \times I_y)$;
`/* Do filtering */`
$F \leftarrow I$;
$m \leftarrow 0$;
$V \leftarrow V^\top$ `// the vertical pass is done on the transposed image`
for $m < M$ **do**
  $\sigma_m \leftarrow \sigma_s \times \sqrt{3} \times 2^{M-(m+1)} / \sqrt{4^M - 1}$;
  $F \leftarrow$ `Recursive1DFilter`(F, H, σ_m);
  $F \leftarrow F^\top$;
  $F \leftarrow$ `Recursive1DFilter`(F, V, σ_m);
  $F \leftarrow F^\top$;
  $m \leftarrow m + 1$;
end

Algorithm 7: `Recursive1DFilter`

Data: Image I, distances D, parameter σ
Result: Horizontally filtered image F

$F \leftarrow I$;
$h, w, c \leftarrow \texttt{shape}(I)$;
`/* Compute the feedback coefficient` V `*/`
$a \leftarrow \exp(-\sqrt{2}/\sigma)$;
$V \leftarrow a^D$; `// the exponent is pixelwise`
`/* Left-to-right filtering */`
$i \leftarrow 1$;
for $i < w$ **do**
  $k \leftarrow 0$;
  for $k < c$ **do**
    $F(:, i, k) \leftarrow F(:, i, k) + V(:, i) \times [F(:, i-1, k) - F(:, i, k)]$;
    $k \leftarrow k + 1$;
  end
  $i \leftarrow i + 1$;
end
`/* Right-to-left filtering */`
$i \leftarrow w - 2$;
for $i \geq 0$ **do**
  $k \leftarrow 0$;
  for $k < c$ **do**
    $F(:, i, k) \leftarrow F(:, i, k) + V(:, i+1) \times [F(:, i+1, k) - F(:, i, k)]$;
    $k \leftarrow k + 1$;
  end
  $i \leftarrow i - 1$;
end

We predict restored patches $\widehat{u}_p$ and gather them into a single $H \times W$ restored image $\widehat{u}$. Formally this aggregation reads

$$\widehat{u} = \frac{\sum_{p \in P} F_p(w_p \widehat{u}_p)}{\sum_{p \in P} F_p(w_p)}, \tag{30}$$

where w_p is a Kaiser window to prevent fusion artifacts, F_p is a function that replaces the p-th patch at its original location in the $H \times W$ image support, and the division is pixelwise. The set P is defined by the patch size and the percentage of overlapping between neighboring patches. In our implementation, we enable the patch decomposition if one of the spatial dimensions among H and W is greater than 600 pixels. If so, we decompose the image into 400×400 tiles with 25% of overlap.

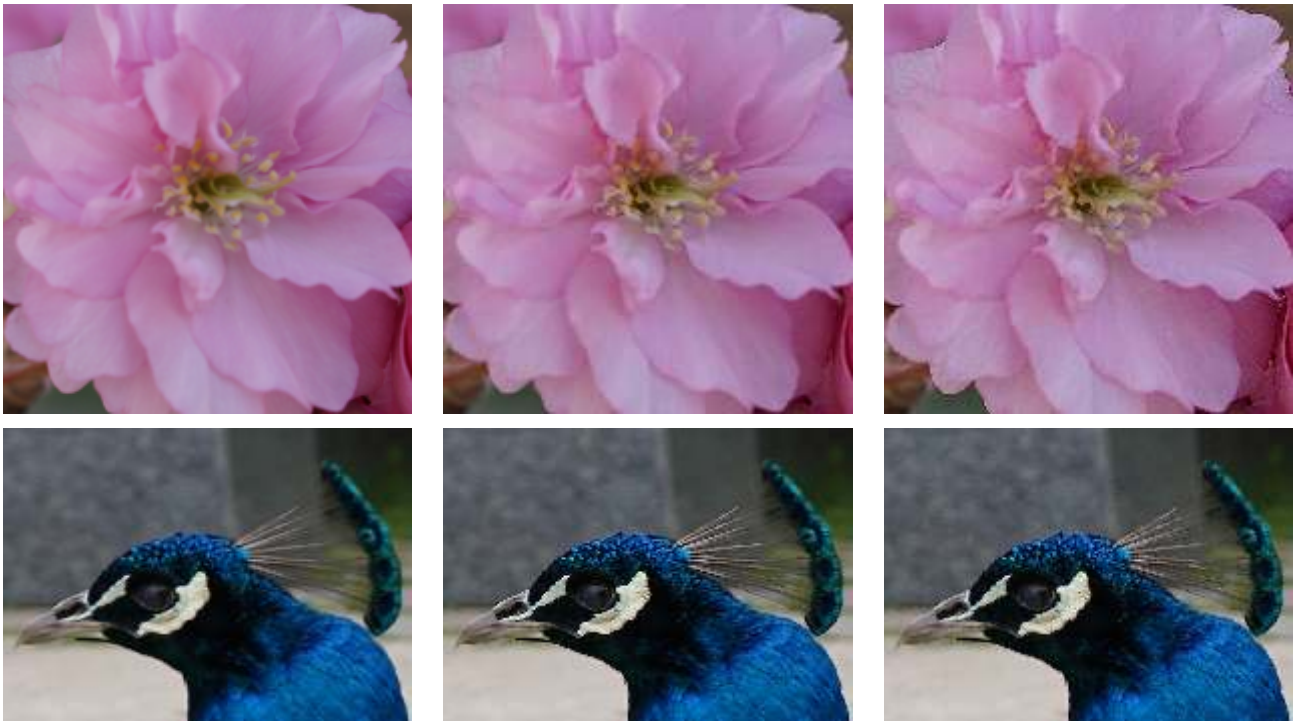
Capturing isotropic blurs. The method detailed above explicitly addresses the removal of anisotropic blurs with different values of the standard deviation along the two principal axes of a Gaussian filter. Yet, this model also accounts for isotropic blurs, such as the defocus kernel in Figure 1, typically captured by a disk-shaped filter [19]. Indeed if the blur is isotropic, the infinite norm of the image gradients along any direction will be *most likely* the same after blur, ultimately resulting in similar values for σ and ρ.

4 Experiments

We test here Polyblur on several real images. We explore restoration of three typical cases: Defocus correction, lens blur compensation and sharpening. Lastly, we contemplate some of its limitations. In all the experiments, we run $N = 3$ iterations of Polyblur with prefiltering enabled, and parameters $\alpha = 6$, $\beta = 1$, $C = 0.362$, $b = 0.468$, $\sigma_s = 2$, $\sigma_r = 0.8$, $M = 3$ and $t = 0.95$.

4.1 Defocus Compensation

In Figure 8 we illustrate the effectiveness of Polyblur for compensating defocus blur. Since defocus blur is often modeled as a disk or a Gaussian filter, the technique of Delbracio et al. is particularly suited for this scenario. We compare Polyblur with the blind deblurring technique of [2], and show that both methods improve the sharpness of the photographs, e.g., the center of the flower and the eye of the peacock, but Polyblur is faster: It runs in a fraction of a second on a GPU and less than 1 second on the CPU of a laptop for this 700×500 sample, whereas the optimization algorithm of [2] runs in about 10 seconds.



(a) Slightly out of focus. (b) Polyblur (blind). (c) Deblurring (blind) [2].

Figure 8: Slight defocus compensation qualitative examples. We compare Polyblur with the multi-purpose blind deblurring technique of [2]. Both methods improve the sharpness of the photographs, e.g., the center of the flower and the eye of the peacock, but Polyblur is significantly faster. The reader is invited to zoom in on a computer screen.

4.2 Lens Blur Correction

In Figure 9 we compare the lens blur correction ability of our method with the results of the non-blind deblurring algorithm of [14] and the blind variant of [15]. We show point-spread function (PSF) compensation results for two JPEG images from [14]. Polyblur, despite being a blind method, improves the sharpness of both examples, especially in the second row where the visual accuracy exceeds that of [14]. We also showcase the versatility of Polyblur for restoring a so-called "historical"

scanned analog photograph from [15] in Figure 10. In this example too, Polyblur achieves a visual result on par with that of the optimization scheme of [15], accurately postprocessing the optics of an old analog camera. Polyblur, and in particular this implementation, when combined with a chromatic aberration compensation technique yields a fast and accurate optical aberration correction, as showcased by a previous work [6].

(a) Aberrated. (b) Polyblur (blind). (c) Deblurring (non-blind) [14].

Figure 9: Optical aberration compensation for two JPEG images from Schuler et al. [14] taken with a Canon 5D Mark II camera and a Canon 50mm f2.8 lens at maximum aperture. We compare Polyblur, a blind deblurring technique, with the non-blind deblurring approach of [14]. Polyblur efficiently compensates the loss of sharpness caused by the lens PSF, despite being blind.

(a) Historical. (b) Polyblur (blind). (c) Deblurring (blind) [15].

Figure 10: Example of a historical photograph used in [15], restored by blind deblurring. Our implementation of Polyblur achieves a visual result comparable to that of Schuler et al. [15]'s blind optimization-based approach, but in a fraction of the running time.

4.3 Sharpening

Sharpening is useful to improve the contrast of images obtained with super-resolution algorithms, which is necessary for personal photography. It may be used to enhance the images obtained with a super-resolution (SR) algorithm. We improve the contrast of two examples obtained with the technique of Lecouat et al. [11], achieving better results than a typical sharpening filter designed to boost the mid-frequencies with a Gaussian mask (we used the implementation of `skimage.filters.unsharp_mask`). We show these images in Figure 11 for a SR result provided by the authors of [11], prior to any sharpening. The version predicted by Polyblur is much sharper than the one obtained with the typical sharpening technique.

(a) SR result of [11]. (b) Polyblur. (c) Unsharp mask.

Figure 11: Examples of sharpened images. The first column shows crops from an SR result of [11] with factor 4, and prior to sharpening. The middle column shows the sharpened results after Polyblur, and the last one the results obtained with `skimage.filters.unsharp_mask`. The images restored with Polyblur are much sharper.

4.4 Limitations

We have shown throughout this presentation that Polyblur is an accurate, yet computationally-efficient, blind deblurring technique applicable to a wide range of realistic scenarios. One of its main merits is to predict an anisotropic Gaussian blur from the gradients of the blurry image, supposing the existence of underlying contrasted edges. This assumption, reasonable for most natural images, may not hold in specific situations such as textures. Figure 12 shows a lens blur correction example in a 50 megapixel photograph. Since the point-spread function spatially varies across the field of view of a lens, we run Polyblur on patches as in [14, 15], with sizes ranging from 100×100 to 800×800. The illustration shows that for the sizes 100×100 and 200×200, the blur estimated on the tree leads to artifacts, whereas for larger patch sizes, some salient edges are preferred by Polyblur, resulting in a more satisfactory sharpening.

Furthermore, Polyblur is very sensitive to noise as shown in Figure 13. We have calibrated the affine parameters C and b with the `CalibratingAffineModel` procedure in Algorithm 3 with the same protocol as in Section 3, but with Gaussian noise level σ_n ranging from 0 (no noise) to 0.03 (medium noise). On the one hand, for $\sigma_n = 0$ and $\sigma_n = 0.01$, a clear correlation between the

(a) 100×100 patches. (b) 200×200 patches. (c) 400×400 patches. (d) 800×800 patches.

Figure 12: Example of failure of Polyblur for deblurring textures. For the smaller patch sizes, Polyblur fails to find salient edges on the tree, which results in artifacts for the 200×200 patches. For larger patches, the problem is alleviated as salient edges are more likely to be found on the patches' supports. The patches in this example do not overlap to highlight the local patch-specific artifacts.

measurements and the linear/affine fit can be seen. Note that the linear model was predicted in Section 2 when noise is omitted. On the other hand, for greater noise levels $\sigma_n = 0.02$ and $\sigma_n = 0.03$, the affine model does not hold anymore, highlighting the lack of robustness of Polyblur to noise.

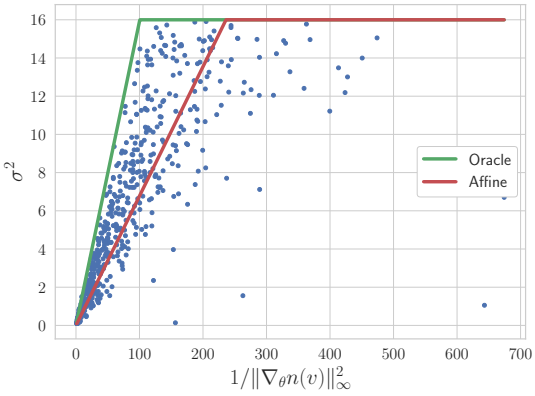
Lastly, Figure 14 shows an image from [17] containing camera shake. One can tell by the saturated light streaks in the blurry image on the left that the blur cannot be captured by any anisotropic Gaussian kernel, and thus cannot be categorized as "mild" blur. The image on the right is the estimate of Polyblur, and is a typical failure case of the method: The blur has only been marginally removed.

5 Conclusion

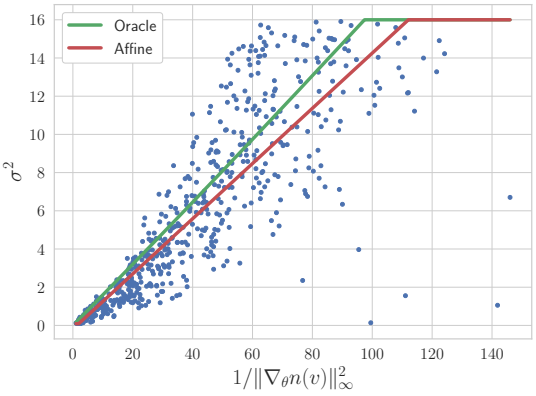
We have presented in this paper a analysis of Polyblur proposed by [5] to address "mild" blurs blind removal. In a first step, a Gaussian kernel accounting for the real blur is quickly computed from the blurry image gradients. In a second stage, an approximate inverse filter to the Gaussian kernel is used to deconvolve the image. The blur is gradually removed by repeating these two stages. Along with an analysis of Polyblur, we provide with this paper two Python implementations based on the Numpy and the Pytorch frameworks. Future work on this algorithm may focus on improving its robustness to texture and noise.

Acknowledgements

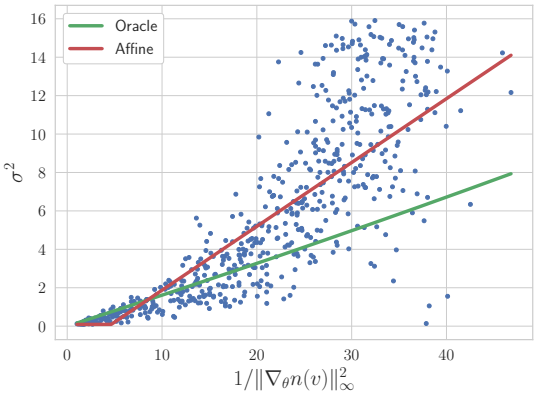
This work was partly financed by DGA Astrid Maturation project "SURECAVI" no ANR-21-ASM3-0002 and Office of Naval research grant N00014-17-1-2552. We thank Bruno Lecouat for providing the images in Figure 11.



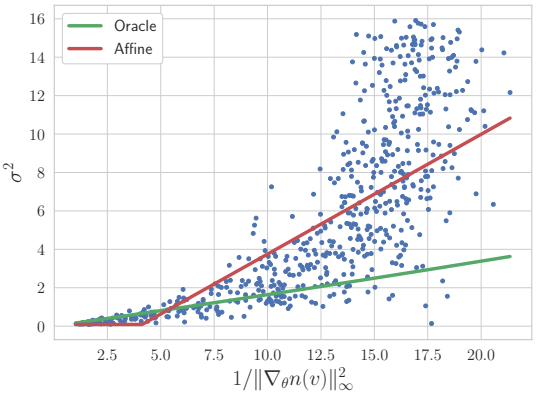
(a) $\sigma_n = 0$.



(b) $\sigma_n = 0.01$.



(c) $\sigma_n = 0.02$.



(d) $\sigma_n = 0.03$.

Figure 13: Calibration curves for four noise levels between σ and $\|\nabla_\theta n(v)\|^{-1}_\infty$, ranging from none to moderate degradation. For $\sigma_n = 0$ and $\sigma_n = 0.01$, we find affine models close to the oracles of Equation (19). Increased noise levels result in important misfits between the samples and the fitted model, highlighting the weakness of Polyblur to important noise levels. The oracle model predicts instead a conservative prediction of the blur, without considering the noisy measurements.

(a) Blurry.

(b) Restored with Polyblur.

Figure 14: A camera shake example from [17], where the blur is clearly not captured by anisotropic Gaussian filters. Polyblur fails to restore the image, and only marginally improves the global contrast.

Image Credits

The blurry images in Figures 1, 8 and 12 are from the authors. Figure 6 was obtained from a test image of the DIV2K dataset. Figure 7 was obtained with the `skimage.data.chelsea` routine. The blurry images in Figures 9 and 10 come from the supplemental materials of Schuler et al. [14, 15]. The SR image in Figure 11 is courtesy of Lecouat et al. [11].

References

[1] J. Anger, G. Facciolo, and M. Delbracio, *Estimating an Image's Blur Kernel Using Natural Image Statistics, and Deblurring it: An Analysis of the Goldstein-Fattal Method*, Image Processing Online (IPOL), 8 (2018), pp. 282–304. https://doi.org/10.5201/ipol.2018.211.

[2] ——, *Blind image deblurring using the ℓ_0 gradient prior*, Image Processing Online (IPOL), 9 (2019), pp. 124–142. https://doi.org/10.5201/ipol.2019.243.

[3] P. Arbelaez, M. Maire, C.C. Fowlkes, and J. Malik, *Contour detection and hierarchical image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 33 (2011), pp. 898–916. https://doi.org/10.1109/TPAMI.2010.161.

[4] S. Boyd and L. Vandenberghe, *Convex optimization*, Cambridge university Press, 2004. ISBN 978-0521833783.

[5] M. Delbracio, I. Garcia-Dorado, S. Choi, D. Kelly, and P. Milanfar, *Polyblur: Removing mild blur by polynomial reblurring*, IEEE Transactions on Computational Imaging (TCI), 7 (2021), pp. 837–848. https://doi.org/10.1109/TCI.2021.3100998.

[6] T. Eboli, J-M. Morel, and G. Facciolo, *Fast two-step blind optical aberration correction*, in European Conference on Computer Vision (ECCV), 2022. http://dx.doi.org/10.48550/arXiv.2208.00950.

[7] E.S. Lopes Gastal and M.M. Oliveira, *Domain transform for edge-aware image and video processing*, ACM Transactions on Graphics (ToG), 30 (2011), p. 69. https://doi.org/10.1145/2010324.1964964.

[8] A. Goldstein and R. Fattal, *Blur-kernel estimation from spectral irregularities*, in European Conference on Computer Vision (ECCV), 2012, pp. 622–635. https://doi.org/10.1007/978-3-642-33715-4_45.

[9] M. Hirsch, C.J. Schuler, S. Harmeling, and B. Schölkopf, *Fast removal of non-uniform camera shake*, in International Conference on Computer Vision (ICCV), 2011, pp. 463–470. https://doi.org/10.1109/ICCV.2011.6126276.

[10] E. Kee, S. Paris, S. Chen, and J. Wang, *Modeling and removing spatially-varying optical blur*, in International Conference on Computational Photography (ICCP), 2011, pp. 1–8. https://doi.org/10.1109/ICCPHOT.2011.5753120.

[11] B. Lecouat, T. Eboli, J. Ponce, and J. Mairal, *High dynamic range and super-resolution from raw image bursts*, ACM Transactions on Graphics (ToG), 41 (2022), pp. 38:1–38:21. https://doi.org/10.1145/3528223.3530180.

[12] J. Pan, D. Sun, H. Pfister, and M-H. Yang, *Deblurring images via dark channel prior*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 40 (2018), pp. 2315–2328. https://doi.org/10.1109/TPAMI.2017.2753804.

[13] L.I. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Physica D, 60 (1992), pp. 259–268. https://doi.org/10.1016/0167-2789(92)90242-F.

[14] C.J. Schuler, M. Hirsch, S. Harmeling, and B. Schölkopf, *Non-stationary correction of optical aberrations*, in International Conference on Computer Vision (ICCV), 2011, pp. 659–666. https://doi.org/10.1109/ICCV.2011.6126301.

[15] ——, *Blind correction of optical aberrations*, in European Conference on Computer Vision (ECCV), 2012, pp. 187–200. https://doi.org/10.1007/978-3-642-33712-3_14.

[16] O. Whyte, J. Sivic, and A. Zisserman, *Deblurring shaken and partially saturated images*, International Journal on Computer Vision (IJCV), 110 (2014), pp. 185–201. https://doi.org/10.1109/ICCVW.2011.6130327.

[17] O. Whyte, J. Sivic, A. Zisserman, and J. Ponce, *Non-uniform deblurring for shaken images*, International Journal on Computer Vision (IJCV), 98 (2012), pp. 168–186. https://doi.org/10.1109/CVPR.2010.5540175.

[18] L. Xu, S. Zheng, and J. Jia, *Unnatural L0 sparse representation for natural image deblurring*, in Conference on Computer Vision and Pattern Recognition (CVPR), 2013, pp. 1107–1114. https://doi.org/10.1109/CVPR.2013.147.

[19] X. Zhu, S. Cohen, S. Schiller, and P. Milanfar, *Estimating spatially varying defocus blur from A single image*, IEEE Transactions on Image Processing (TIP), 22 (2013), pp. 4879–4891. https://doi.org/10.1109/TIP.2013.2279316.