Published in Image Processing On Line on 2023–07–16.
Submitted on 2023–01–16, accepted on 2023–06–18.
ISSN 2105–1232 © 2023 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.2023.460

Implementing Handheld Burst Super-resolution

Jamy Lafenetre, Gabriele Facciolo, Thomas Eboli

Université Paris-Saclay, ENS Paris-Saclay, Centre Borelli, France

Communicated by P. Musé and T. Ehret *Demo edited by* J. Lafenetre and T. Eboli

Abstract

Nowadays, smartphone cameras capture bursts of raw photographs whenever the trigger is pressed. These photos are then fused to produce a single picture with higher quality. This paper details the implementation of the method “Handheld Multi-Frame Super-Resolution algorithm” by Wronski et al. (used in the Google Pixel 3 camera), which performs simultaneously multi-image super-resolution demosaicking and denoising from a burst of images. Hand tremors during exposure cause subpixel motions, which combined with the Bayer color filter array of the sensor results in a collection of aliased and shifted raw photographs of the same scene. The algorithm efficiently aligns and fuses these signals into a single high-resolution one by leveraging the aliasing to reconstruct the high-frequencies of the signal up to the Nyquist rate of the sensor. This approach yields digitally zoomed images up to a factor of 2, which is the limit naturally set by the sensor pixel integration. We present an in-depth description of this algorithm, along with numerous implementation details we have found to reproduce the results of the original paper, whose code is not publicly available.

Source Code

The source code for reproducing the results presented in this paper, and an online demo, are available at the associated web page[1]. The source code is also available in the first author’s GitHub page[2]. It is implemented in Python with Numba support for fast GPU-based computations.

Keywords: burst processing; demosaicking; super-resolution; raw photographs

1 Introduction

The smartphone cameras are nowadays the preferred devices to take photographs, much more than the traditional digital cameras [6]. Despite the difference in imaging sensor sizes smartphone cameras attain comparable image quality on the resulting sRGB images. This is easily explained by the growing performance of the embedded image processing algorithms that nowadays attain acceptable

[1]https://doi.org/10.5201/ipol.2023.460
[2]https://github.com/Jamy-L/Handheld-Multi-Frame-Super-Resolution

```
Algorithm 1: HandheldBurstSuperResolution
Data: Coarse raw frames $\{J_1, \dots, J_N\}$, zoom $s$, noise model $(\alpha, \beta)$
Result: Fine RGB image $I$
1  $H, W \leftarrow$ Shape($J_1$);
2  num $\leftarrow$ Zeros($sH, sW, 3$);
3  den $\leftarrow$ Zeros($sH, sW, 3$);
4  $R_c \leftarrow$ Zeros($H/2, W/2$);
   // Aggregating all the frames but the reference one.
5  for $n = 2, \dots, N$ do
       // Registration
6      $V_n \leftarrow$ Registration($J_1, J_n$);
       // Robustness computation
7      $r_n \leftarrow$ ComputeRobustness($J_1, J_n, V_n$);
       // Accumulated robustness update
8      $R_c \leftarrow R_c + r_n$;
       // kernel estimation
9      $\Omega_n \leftarrow$ ComputeKernelCovariance($J_n, (\alpha, \beta)$);
       // Fusion
10     num, den $\leftarrow$ Accumulation ($J_n, \Omega_n, r_n, V_n, s$, num, den) ;
11 end
   // Aggregate the reference frame at the end, taking into the areas in single-frame mode.
12 $\Omega_1 \leftarrow$ ComputeKernelCovariance($J_1, (\alpha, \beta)$);
13 num, den $\leftarrow$ AccumulationReference($J_1, \Omega_1, s$, num, den, $R_c$);
   // Normalization of the predicted image.
14 $I \leftarrow$ num/den;
```

running times and accuracy for in-the-wild photography, all that while running on devices that fit in our pockets. These algorithms compensate for the poor lenses and very small sensors of smartphones by using multi-image restoration algorithms that fully exploit the continuous acquisition capabilities of these devices. In particular, manufacturers are increasingly looking for effective and lightweight multi-frame super-resolution (SR) algorithms, for better zooms.

We present an implementation of the raw burst SR algorithm by [25], which is used for the digital zoom of the Google Pixel 3 phone. This approach yields a high-resolution RGB image from a burst of raw photographs taken with a handheld camera. The hand tremors cause camera pose changes during the acquisition; What seems to be a bane at first glance is actually a boon. Because of the integration on the sensor, each frame is most likely aliased–that is some of the high-frequency content beyond the Nyquist rate is injected within the lower-frequency content [18, Sec. 3.1.2]. Pose changes allow to sample different positions of the aliased image in each frame, and thus many missing high frequencies may be accurately recovered from the combination of the coarse measurements [24] up to a factor 2. Beyond this resolution gain, the problem is more ill-posed and requires image priors to observe further resolution improvements [1]. This is however, beyond the scope of this paper.

Formally, the approach builds a $sH \times sW$ high resolution (HR) color image I from N raw aliased low resolution (LR) frames $J_1, \dots, J_N$, each one of size $H \times W$. The scalar s is the zoom factor specified by the user in $[1, +\infty)$ (even though beyond 2, little improvements are to be expected). The method follows a classical pipeline: (1) it aligns $N-1$ frames to a reference one, for instance the first one J_1, and (2) it merges the registered frames into I with respect to some signal-adaptive aggregation weights computed on the fly. The global architecture of the method is shown in Algorithm 1.

The particularity of this technique lies in how the frame fusion is achieved. The merging weights have two distinct components that are equally important. The first one accounts for the geometric features of the image by means of local structure tensors. This allows to adapt the aggregation weights to account for flat areas, edges or corners, preventing blurring high-frequency details while denoising the flatter regions. The second component is an improved deghosting mask designed to filter out elements of the LR frames that are misaligned in a noise-aware fashion.

In what follows, we will present the alignment and fusion strategy of [25] with our implementation details to fill the elements eluded in the presentation of the original paper.

2 Method

In this section we described the two main stages of the approach. In Section 2.1, we detail the registration step that predicts the optical flows between the reference frame and the remaining "moving" frames of the burst. In Section 2.2, we describe how the frames are merged with the help of the previous flows into an HR image.

Algorithm 2: `Registration`

```
Data: Reference $J_1$, Moving $J_n$, tile size $T$, reference radius $R$
Result: Patchwise flow $V_n = \{V_n^{(1)}, V_n^{(2)}, \dots, V_n^{(P)}\}$
// Compute a high resolution version of the images
1  $G_1 \leftarrow$ ComputeGrayscaleImage($J_1$);
2  $G_n \leftarrow$ ComputeGrayscaleImage($J_n$);
// Decompose the reference image into a collection of $P$ tiles. $P_1$ refers to the non-overlapping
   patches of size $T \times T$ that are a partition of $J_1$.
3  $P_1 \leftarrow$ ToTiles($G_1, T$);
4  $p \leftarrow 1$;
// For each tile, predict the local parametric flow
5  for $p_1 \in P_1$ do
       // Predicts a patchwise subpixel translation with BM.
6      $V_n^{(p)} \leftarrow$ MultiScaleBlockMatching($p_1, G_n$);
       // Predicts a refined patchwise subpixel translation with 3 ICA iterations.
7      $i \leftarrow 0$;
8      while $i < 3$ do
9          $V_n^{(p)} \leftarrow$ ICA($p_1, G_n, V_n^{(p)}$);
10         $i \leftarrow i + 1$;
11     end
12     $p \leftarrow p + 1$;
13 end
```

2.1 Registration

The first stage consists in aligning all the coarse frames J_n ($n = 2, \dots, N$) over the reference one J_1 with an accurate sub-pixel optical flow. This is crucial to leverage aliasing across the frames, and thus achieving satisfactory SR. To do so, Wronski et al. [25] partition the reference frame J_1 into P patches and estimate an optical flow $V_n = \{V_n^{(1)}, V_n^{(2)}, \dots, V_n^{(P)}\}$ between each patch and each "moving" frame J_n ($n = 2, \dots, N$). This is achieved in the original paper by a two-stage strategy.

First, a *coarse* flow with a multi-scale block-matching (BM) algorithm is predicted. The coarse flow is then used as initialization for "three iterations of the Lucas-Kanade (LK) algorithm", to quote the authors, which yields a *refined* flow with sub-pixel accuracy.

Note that in this section we will not detail any registration strategy as the BM algorithm is covered in detail in [21] and the LK iterations are detailed in [22]. Algorithm 2 presents a pseudo-code for the registration module assembling the two stages, which we now survey individually.

Coarse registration: multi-scale block matching. A coarse estimate of the flow is estimated with a multi-scale BM algorithm [13] that finds discrete translation flows between patches of two different frames. The BM prediction is computed using a coarse-to-fine strategy applied on a Gaussian pyramid of 4 levels. At each level patchwise translations are estimated, then used to initialize the next finer stage. To facilitate the flow upscaling between stages, the pyramid is built in such a way that the tiles are split into an integer number of subtiles to reach a finer pyramid level.

Formally, for a given pixel location (x, y) in the reference image, at a given level of the pyramid, we extract a $T \times T$ local reference patch p_1 in J_1 and a corresponding larger tile t_n at the same location in the secondary image J_n $(n = 2, \dots, N)$. The second tile is necessarily made larger, with size $(T + 2R) \times (T + 2R)$, where R is the search radius and whose center is $(x, y) + U$, to predict the translation V. The 2D vector $U = (U_x, U_y)$ is $(0, 0)$ at the coarsest scale, and for all the other scales it is initialized from the previous scale. For a given patch p_1, the flow is initialized by considering the 3 closest tiles on the coarsest pyramid level. The patch p_1 is, by construction, fully contained within one of these tiles. The respective flows of the 3 tiles are separately applied to p_1 and associated with the ℓ_1 error between p_1 and the matching position in t_n. The flow of p_1 is the flow, among the 3 candidates, that minimizes the error. In this setting, the flow U comprises all the motions that were estimated up to the current level of the pyramid, whereas V is the "innovation" brought by the new level, found with

$$\min_{V \in \mathcal{R}} \sum_{(x,y) \in p_1} \| p_1(x, y) - t_n((x, y) + U + V) \|_\ell^\ell, \tag{1}$$

for a given ℓ-norm and $\mathcal{R} = \{-R, -R + 1, \dots, R\}^2$. In practice, at the three coarsest scales we set ℓ to be 2, but for the finer one Hasinoff et al. [13] suggest using the ℓ_1 norm to avoid smoothed results favored by the ℓ_2 norm. The implementation details of this BM scheme are presented in [21]. Unlike Hasinoff et al. , our merging method is not sensitive to flow discontinuities: this is why we can split the reference image into non-overlapping tiles, which yields an improved execution time.

Fine registration: ICA iterations. The flow computed in the previous BM step is precise up to one pixel, whereas we need *sub-pixel* accuracy to achieve a satisfactory super-resolution. Yet, the discrete flow predicted by BM is a fast initialization for a subsequent subpixel refinement. The original paper by Wronski et al. only states that "three iterations of the Lucas-Kanade algorithm" were applied, with no further details. It is thus hard to infer what the authors actually did, but we posit that they used an inverse compositional algorithm (ICA), a cheaper, yet at least as accurate variant of LK [2]. Starting from the BM initial guess of the translation for V_n, the ICA method [2] solves the problem

$$\min_{dV_n \in \mathbb{R}^2} \sum_{(x,y) \in p_1} \| p_1(dV_n(x, y)) - J_n(V_n(x, y)) \|_2^2, \tag{2}$$

where V_n is the current estimate of the flow and dV_n is the subpixel refinement predicted by ICA. The new estimate of the flow becomes: $V_n \leftarrow V_n + dV_n$. We estimate dV_n for each tile returned by the BM algorithm. After 3 ICA iterations on each one of the P patches, we have the final estimate of the subpixel patchwise translation V_n between J_1 and J_n $(n = 2, \dots, N)$. The ICA algorithm is detailed in [22].

Note that in the original paper [25], the authors may have run ICA after block-matching at each stage of the pyramid, the text in the paper is not very clear. Yet, our approach of using ICA only at the finer level of the pyramid yields satisfactory-enough results in practice.

CFA images pre-processing. The registration approaches such as BM and ICA are valid on grayscale images, and subpixel image registration methods are most accurate when applied to alias-free images. However, since the inputs of the SR algorithm are aliased CFA frames, a pre-processing stage is needed before registration. The manuscript by Wronski et al. is not really accurate on this point, and we posit that a cheap and fast possibility that may have been actually considered by the authors is to compute decimated grayscale images by averaging each 2×2 Bayer quad, as in a previous work by the same team [13]. The resulting images are twice smaller but grayscale (thus aligned with running a registration algorithm), de-aliased/denoised thanks to the averaging of the pixels, and computations are faster by running on smaller images. This is a totally fine approach for quick computing of optical flow on a flagship smartphone released on 2018. For a pseudo-code of this approach, please refer to [21, Algorithm 1].

Yet, in this draft and for the sake of the companion demo that runs on a workstation, we explore another option proposed in [4, Chapter 8] to convert the raw CFA images into grayscale images of the *same* resolution as the raw ones, which should lead to superior results in the end. It consists of filtering the frequencies of the Bayer images above $\pi/2$ with a perfect low-pass filter in the Fourier domain. The associated cost is two fast Fourier transforms (FFT) per frame to be aligned. The same processing is applied in the case of CFA images or to aliased grayscale images for the case of monochromatic sensors. The corresponding preprocessing pseudo-code is shown in Algorithm 3. Note that if required, the interested reader can replace our handling of CFA images for alignment with the Bayer quad averaging strategy of [13] to be closer to the original implementation of Wronski et al.

Algorithm 3: `ComputeGrayscaleImage`

```
   Data: Grayscale or mosaicked $H \times W$ image $J$
   Result: Grayscale image $G$
1  if $J$ is grayscale then
2      $G \leftarrow J$;
3  else
       // Assuming the (0,0) frequency is in the center, sets to 0 the coefficients outside $[-\pi/2, \pi/2]^2$.
4      $H, W \leftarrow$ Shape($J$);
5      $G \leftarrow$ FFT2($J$);
6      $G[: H/4, :] \leftarrow 0$;
7      $G[3H/4 :, :] \leftarrow 0$;
8      $G[:, : W/4] \leftarrow 0$;
9      $G[:, 3W/4 :] \leftarrow 0$;
10     $G \leftarrow$ IFFT2($G$);
11 end
```

Validation of the pre-processing. Both BM and ICA algorithms can separately be run on either (i) FFT-filtered images [4, Chapter 8], (ii) demosaicked images [20], or (iii) the decimated grayscale images obtained by averaging the 2×2 Bayer quads. To compare these approaches we have generated a synthetic burst of images from which 2280 aligned patches were extracted, we then evaluate by comparing the quadratic error of the predicted flows. To take into account the potential difference of resolution between gray images, we have adjusted the size T of the patches so that all the gray images could be split into the same number of patches. Figure 1 shows registration results with respect to the number of ICA iterations.

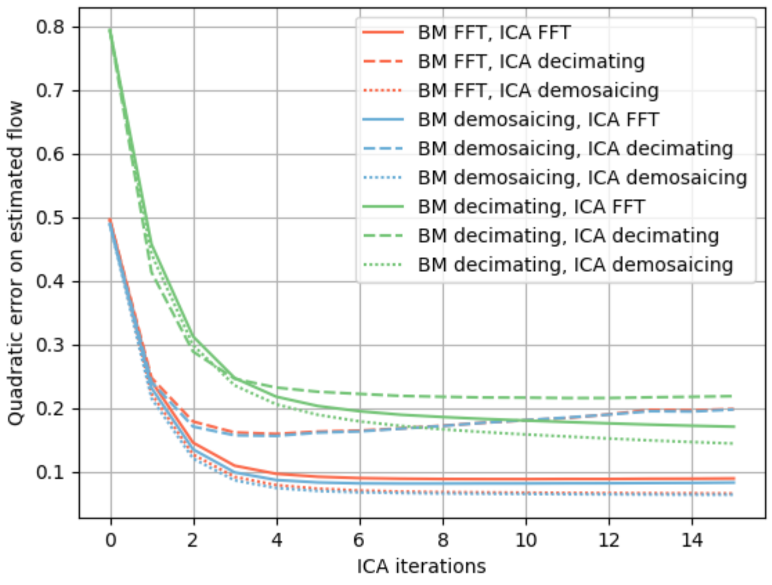


Figure 1: Comparison of the registration error for different methods. We compare the two stages of the presented alignment pipeline for three different raw-to-grayscale image conversion strategies: Or the decimated quad Bayer image [13], or the full-resolution images obtained via demosaicking or FFT perfect low-pass filtering [4]. Running the two algorithms sequentially on the FFT-filtered images, with 3 ICA iterations, is the optimal strategy.

We first observe that the choice of the preprocessing for BM is crucial: the decimation method is outperformed by the FFT-filtering and demosaicking variants, by a gap that cannot be compensated using subsequent ICA iterations. Our second observation is that the asymptotic error of the ICA algorithm nearly doubles when running on a decimated image, compared with the FFT-filtering and demosaicking methods. Last but not least, demosaicking and FFT filtering achieve marginally the same results with converging values reached after three iterations (about 0.1 pixel of error), which validates the choice of the "three LK iterations" in [25]. The FFT-based strategy is however faster than the demosaicking one in practice since applying a low-pass filter to an image is in general a much easier problem than (well) interpolating the missing color components of a mosaicked image. This concludes the justification of running BM and just three ICA iterations on the images filtered via the FFT strategy in this context.

Limits on the registration pipeline. The registration strategy of Wronski et al. has the same limitations as any local optical flow algorithm such as LK. Because of the aperture problem next to the edges, the structure tensor is likely to be ill-conditioned or, even worse, not-invertible. This may lead to unstable flow estimates in these regions [2]. On flat areas, the alignment problem becomes ill-posed as both eigenvalues of the structure tensor approach 0. The flow may therefore be inaccurate. Yet, since in that case we are matching roughly uniform patches, fusion artifacts are unlikely.

2.2 Fusion

Once the frames $J_2, \ldots, J_N$ are registered with J_1 with sub-pixel accuracy, we may merge them to predict an image I with resolution s times finer than that of J_1. A first approach would be to splat

Algorithm 4: `Accumulation`

```
Data: Frame $J_n$, kernels $\Omega_n$, robustness map $r_n$, patch-wise flow $V_n$, zoom $s$, numerator image num, denominator image den
Result: Updated numerator and denominator of the accumulator: num and den
// Main loop over the pixels of the fine grid
for $(x, y) \in num$ do
    // Corresponding location on $J_1$'s coarse grid
    $(x_1, y_1) \leftarrow (x, y)/s$;
    // Projection from $J_1$'s to $J_n$'s coarse grid
    $(x_n, y_n) \leftarrow V_n(x_1, y_1)$;
    // Fetch the 4 closest pre-computed covariance matrices
    for $(\hat{x}_n^i, \hat{y}_n^i) \in \mathcal{N}_2(x_n, y_n)$ do
        $\Omega^i \leftarrow$ Fetch$(\Omega_n, (\hat{x}_n^i, \hat{y}_n^i))$;
    end
    // Interpolate to the desired position based on the known Omegas
    $\Omega \leftarrow$ BilinearInterpolation$(\Omega^1, \Omega^2, \Omega^3, \Omega^4, x_n, y_n)$;
    for $(u, v) \in \mathcal{N}_3(x_n, y_n)$ do
        // Build on-the-fly the merge kernel with $(u, v)$ back-projection on $I$'s fine grid
        $d \leftarrow (u, v) - (x_n, y_n)$;
        $w \leftarrow \exp\left(-d^\top \Omega^{-1} d/2\right)$;
        // Look which color component will be added
        $c \leftarrow$ GetLocalChannel$((u, v))$;
        // Do the aggregation in the image and accumulator
        num$(x, y, c) \leftarrow$ num$(x, y, c) + r w J_n(u, v)$;
        den$(x, y, c) \leftarrow$ den$(x, y, c) + r w$;
    end
end
```

all the observations from the N images on a same continuous space via kernel regression, the samples could then be resampled at the desired resolution as proposed by Takeda et al. [23]. This is however prohibitive as a much larger intermediate image would be needed to store the splatted samples just to estimate a smaller HR image. Another more effective option to filter irregularly sampled data is the normalized convolution (NC) proposed in [14]. We invite the interested readers to also read the portions of the thesis [4, Chapter 9], where the general principles and intuitions for applying the NC to a collection of irregular samples in the context of image SR are explained.

However, in our setting the observations are organized as N collections of $T \times T$ regular grids, whose positions are known relatively to one set as the reference. In this context, Wronski et al. [25] use a memory-efficient variant of NC that sequentially aggregate the contributions in the final HR image I with locally-adaptive weights. Formally, the merge they propose reads

$$I(x, y) = \frac{\sum_{n=1}^{N} r_n(x, y) \sum_{(u,v)\in\mathcal{N}_n^3} w_n\left(x, y, u, v\right) J_n\left(u, v\right)}{\sum_{n=1}^{N} r_n(x, y) \sum_{(u,v)\in\mathcal{N}_n^3} w_n\left(x, y, u, v\right)}, \tag{3}$$

where $\mathcal{N}_n^3$ is a neighborhood of size 3×3 around the (x, y) location projected on the grid of the image J_n (more details in what follows). The notations are made concise for the sake of clarity. The summation over N represents the accumulation through *time*. The summation over the coordinates (u, v) in $\mathcal{N}_n^3$ aggregates the 3×3 pixels in J_n that are *spatially* the closest to (x, y). All the sums are weighted by robustness and fusion weights r_n and w_n. We will detail each weight in what follows.

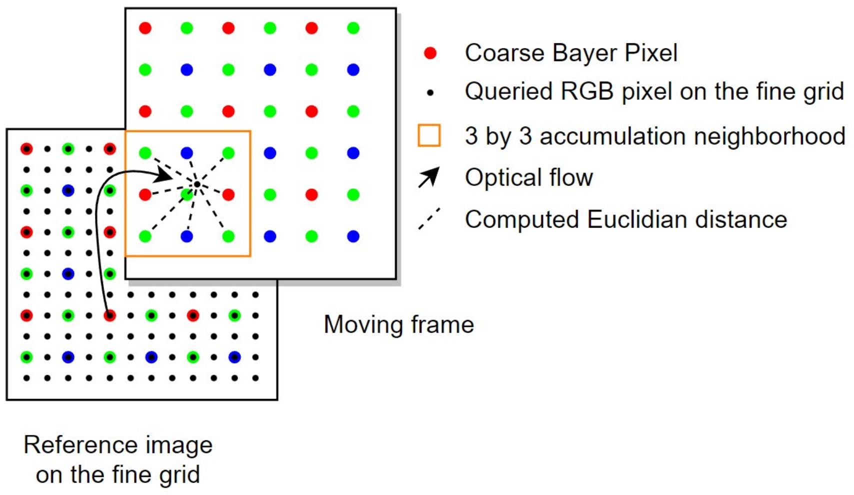

Coarse Bayer Pixel
Queried RGB pixel on the fine grid
3 by 3 accumulation neighborhood
Optical flow
Computed Euclidian distance
Moving frame
Reference image on the fine grid

Figure 2: Logic of the accumulation for one frame. The optical flow is used to determine the queried position in the moving frame, and to compute the distance with the 9 closest pixels.

In what follows we drop the reference to (x, y) in the notation $\mathcal{N}_n^3$ for the sake of clarity. Figure 2 illustrates the fusion scheme for a moving frame and the reference. The corresponding pseudo-code is shown in Algorithm 4.

Note that Algorithm 1 features two fusion algorithms: `Accumulation` and `AccumulationReference` (respectively Algorithm 4 and Algorithm 11), the latter being a variant of the former that we will motivate and detail later in Section 2.3. We made this choice to focus in the current section on the merging scheme, and dedicate Section 2.3 to describing a minor modification when merging the reference frame.

Interpreting the merge equation. For each LR frame J_n, we look where the queried HR pixel (x, y) falls and accumulate the closest 3×3 neighborhood. Thus, each frame contributes 9 weighted observations for generating the HR pixel at (x, y). This strategy allows to run the fusion algorithm in a sequential manner ensuring that only one frame is loaded at the time, thus fixing the memory footprint of the algorithm no matter the burst size N. This is optimal for mobile devices. On larger devices, like a workstation, a fully parallel strategy where several frames are processed in parallel may be preferred if the memory usage is not an issue.

2.2.1 Structural Weight

This fusion weight w_n is defined as the exponential of a distance between the pixel locations (x, y) on the final HR grid, and (u, v) on one of the N LR grids

$$w_n(x, y, u, v) = \exp\left(-\frac{1}{2} d^\top \Omega^{-1} d\right), \tag{4}$$

where Ω is the covariance matrix defining the metric between the two locations, and d is the vector of differences between the two locations. In the context of Equation (3), the latter quantity reads generally

$$d = \begin{bmatrix} x - u \\ y - v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - V_n^{-1}\left(\begin{bmatrix} x_n \\ y_n \end{bmatrix}\right), \tag{5}$$

where (u, v) is the location (x_n, y_n) in the coarse image J_n $(n = 1, \dots, N)$ brought back to the fine grid I with the patch-wise flow V_n computed in the previous section. Euclidean distances are left

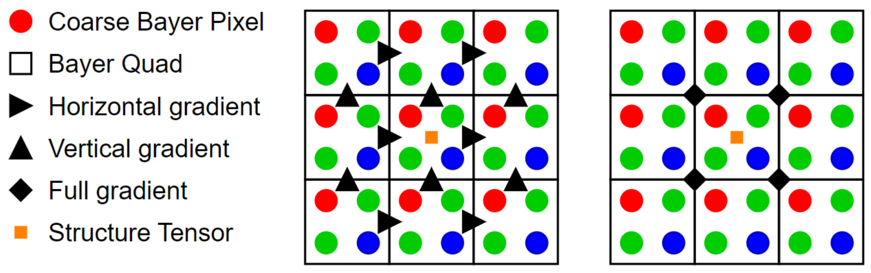


Figure 3: Computation of one structure tensor from a 6×6 coarse neighborhood of Bayer pixels further grouped into 3×3 grayscale quad pixels. The 9 gray pixels are used to compute 6 vertical and 6 horizontal intermediate gradients via finite difference (center). These values are averaged into 4 more robust "full" gradient estimates that are finally used to estimate the structure tensor on the central location of this neighborhood (right).

unchanged by rigid optical flows, allowing us to compute them directly within the moving frame, thus saving the expense of projecting back every position on the reference frame. Formally, it reads

$$d = \begin{bmatrix} x \\ y \end{bmatrix} - V_n^{-1}\left(\begin{bmatrix} x_n \\ y_n \end{bmatrix}\right) = V_n\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) - \begin{bmatrix} x_n \\ y_n \end{bmatrix}. \tag{6}$$

In Equation (4), the inverse of the matrix Ω defines a geodesic that indicates when we cannot integrate through a line to preserve the edges [23]. Note that in the simple case where Ω is the identity matrix, Thibaud Briand showed [4, Chapter 9] that Equation (4) corresponds to a fast approximation of an accurate but expensive spline-based interpolation strategy for super-resolution. The definition of the covariance matrix Ω is the core of this method. We now derive the different steps to compute it locally by taking into account the geometry of the area via the structure tensor and the noise model.

Computing the local structure tensors. Besides the edge case corresponding to the identity, to save computations, the matrices Ω are estimated from low-resolution grayscale versions of the input images. Wronski et al. first compute a version of the Bayer image by averaging the 2×2 RGGB quads, thus resulting in a grayscale G_n version of J_n at half the resolution. We posit this is done to speed-up the computations since in the original paper the authors must process a burst of 12 Megapixel frames as fast as possible. Assuming that the matrices Ω smoothly vary on the coarse grids J_n, working on a 3 Megapixel image instead and estimating Ω on J_n via bilinear interpolation of the coarser grid is indeed an acceptable fast strategy. The logic of the interpolation is illustrated in Figure 6.

We use the structure tensors computed on 3×3 grayscale neighborhoods to shape the kernel with respect to the image structure. We start by computing the vertical and horizontal gradients in the *grayscale* image with the finite difference filters $[1, -1]$ and $[1, -1]^\top$ at six possible locations for each direction within the 3×3 coarse pixels neighborhood. This results in six horizontal gradients and six vertical gradients, estimated at different positions. Figure 3 shows the locations where the one-dimensional gradients are computed. By averaging the two closest vertical gradients and the two closest horizontal gradients around each of the four corners of the central gray pixel, we obtain four different more robust gradient vectors. From these four values, we build the local structure tensor $S(x_n, y_n)$ defined as

$$S(x_n, y_n) = \sum_{i=1}^{4} \begin{bmatrix} [\nabla_x G_n]_i [\nabla_x G_n]_i & [\nabla_x G_n]_i [\nabla_y G_n]_i \\ [\nabla_x G_n]_i [\nabla_y G_n]_i & [\nabla_y G_n]_i [\nabla_y G_n]_i \end{bmatrix} = B^\top \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} B. \tag{7}$$

The summation over the four gradient estimates guarantees that $S(x_n, y_n)$ is rank 2 except on degenerated cases (constant image). In the previous equation for the gradients $\nabla_x G_n$ and $\nabla_y G_n$, and in what follows for S, B, λ_1 and λ_2 we drop the dependency on (x_n, y_n) for conciseness. Since S is symmetric, we only need to compute 3 entries for each pixel i ($i = 1, \dots, 4$).

We noticed after having implemented our code that the lead author of the original article had further detailed in a blog[3] that the gradients were computed diagonally using the Roberts cross operator. Here we connect the Roberts cross with our approach. Let ∇_{0° and ∇_{90° denote the vertical and horizontal gradient operators we use

$$\nabla_{0^\circ} = \frac{1}{4}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad \text{and} \quad \nabla_{90^\circ} = \frac{1}{4}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}. \tag{8}$$

Similarly, let ∇_{45° and ∇_{135° denote the two diagonal gradient operators featured in the Robert cross strategy

$$\nabla_{45^\circ} = \frac{1}{2\sqrt{2}}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{and} \quad \nabla_{135^\circ} = \frac{1}{2\sqrt{2}}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{9}$$

Remark that $\nabla_{45^\circ} = (\nabla_{0^\circ} + \nabla_{90^\circ})/\sqrt{2}$ and $\nabla_{135^\circ} = (\nabla_{90^\circ} - \nabla_{0^\circ})/\sqrt{2}$ are related by a 45° rotation matrix. Our approach thus boils down exactly to the same eigenvalues for the structure tensor as if they were computed with the Roberts cross as discussed in the blog, but the eigenbasis B is simply rotated by 45°. Our implementation thus ultimately steers the kernels as in [25].

The local geometry coefficients. The eigendecomposition of S yields the rotation matrix B and the sorted non-negative eigenvalues λ_1 and λ_2 whose values are descriptors of the presence of a corner, an edge or a flat region, featured for instance in the Harris corner detector [12]. Yet, because the gradients are computed at a very low resolution from aliased and noisy images, these eigenvalues are not totally reliable and should be regularized before being used in image processing algorithms [23]. Wronski et al. perform regularization based on two intermediate quantities A and D accounting respectively for the anisotropy of the local structure on a ratio of λ_1 and λ_2, i.e., are we on an edge or corner/flat region, and the importance of noise over detail, that quantifies the presence of noise based on a threshold set on λ_1

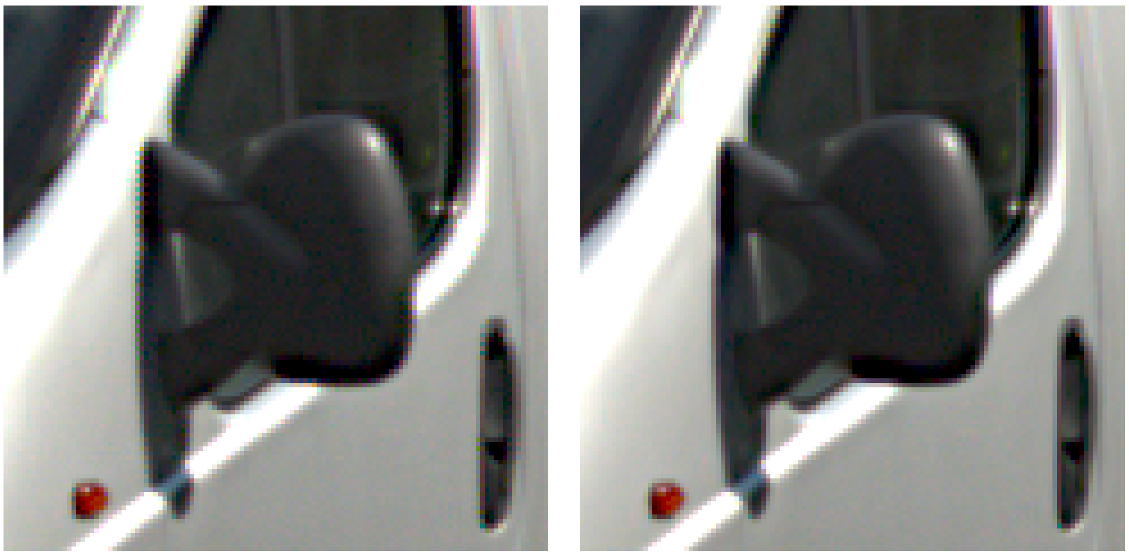
$$A = 1 + \sqrt{\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}} \;\in\; [1, 2], \tag{10a}$$

$$D = \max\left(\min\left(1 - \frac{\sqrt{\lambda_1}}{D_{tr}} + D_{th}, 1\right), 0\right) \;\in\; [0, 1]. \tag{10b}$$

In the previous equations, A measures the anisotropy of the local geometry described by λ_1 and λ_2. It is 2 for pure edges and 1 for pure flat regions or corners. When D is 1, noise dominates over signal whereas D set to 0 is the opposite situation. The importance of noise is carefully handcrafted in the threshold and intercept values D_{tr} and D_{th}. We will use A and D later in this presentation to steer the weighting kernels w_n.

Variance stabilization transform. During the reimplementation of this algorithm, we noted that the dependence on the brightness of the noise caused by the shot noise component was adversarial with the computation of D. Indeed, computing D consists in thresholding the largest eigenvalue λ_1 with two parameters handset to measure the ratio of the signal variations over the noise level. Yet, λ_1 is agnostic to the local brightness level, it only depends on its variations, whereas the noise level is a direct function of the average local brightness. It is thus hard to set a general threshold for all brightness levels. The original paper does not mention this point at all but we assume the authors had to face at some point the issue of varying noise depending on the brightness level.

[3] www.bartwronski.com/2021/02/28/computing-gradients-on-grids-forward-central-and-diagonal-differences/



(a) Isotropic kernels. (b) Anisotropic kernels.

Figure 4: Example of image demosaicking illustrating the advantage of using anisotropic stretched kernels in the case of edges. Isotropic kernels with $k_{shrink} = k_{stretch} = 1$ in (a) cause zipper artifacts, that are less prominent when stretching kernels along edges with $k_{stretch} = 4$ and $k_{shrink} = 2$ in (b).

Without any indication from the paper, we use the general Anscombe transform (GAT) for the Poissonian-Gaussian noise [17, Equation (3)] prior to computing the images G_n ($n = 1, \dots, N$). The GAT is usually used for adapting realistic noise distributions to a Gaussian denoising with tresholds tuned for a single noise standard deviation, and reads

$$\mathrm{GAT}(J) = \frac{2}{\alpha}\sqrt{\alpha J + \frac{3}{8}\alpha^2 + \beta}, \tag{11}$$

where (α, β) is the noise curve provided by the EXIF metadata or computed with the protocol of [9], for instance. After applying the GAT, we observe that setting a single value for D_{th} and D_{tr} is enough to get homogeneously denoised flat regions everywhere in the image. Note that we do not need the inverse of the GAT since the images G_n are only used for computing noise-corrected image gradients.

Shaping the merge kernels. The formulation based on the structure tensor allows to reshape the kernels based on the local geometry. The kernel should thus be isotropic when A is close to 1 ($\lambda_1 \approx \lambda_2$), and the support depends on whether we should preserve details with as small standard deviation k_{detail} or denoise with larger value $k_{detail} \times k_{denoise}$, i.e., D is respectively 0 or 1. However if A is close to 2 ($\lambda_1 \gg \lambda_2$), we should accumulate only along the edge by stretching the kernel in that direction and shrinking its support across the edge.

This choice is motivated by the need to attenuate zipper artifacts, which are prone to appear on sharp edges of strong contrast as seen in Figure 4. We observe in practice that the aperture problem drastically reduces the samples density on edges regions, therefore hindering the use of narrow isotropic kernels. To compensate the absence of samples in the close radius of a queried point, the kernel is stretched along the edge in order to fetch samples that are relevant although distant.

To account for this last case, Wronski et al. magnify the anisotropic shape of the kernel by introducing two scalars $k_{stretch}$ and k_{shrink} whose roles are respectively to manually amplify and reduce the standard deviation along and across the edges. To summarize, the desired kernel should be:

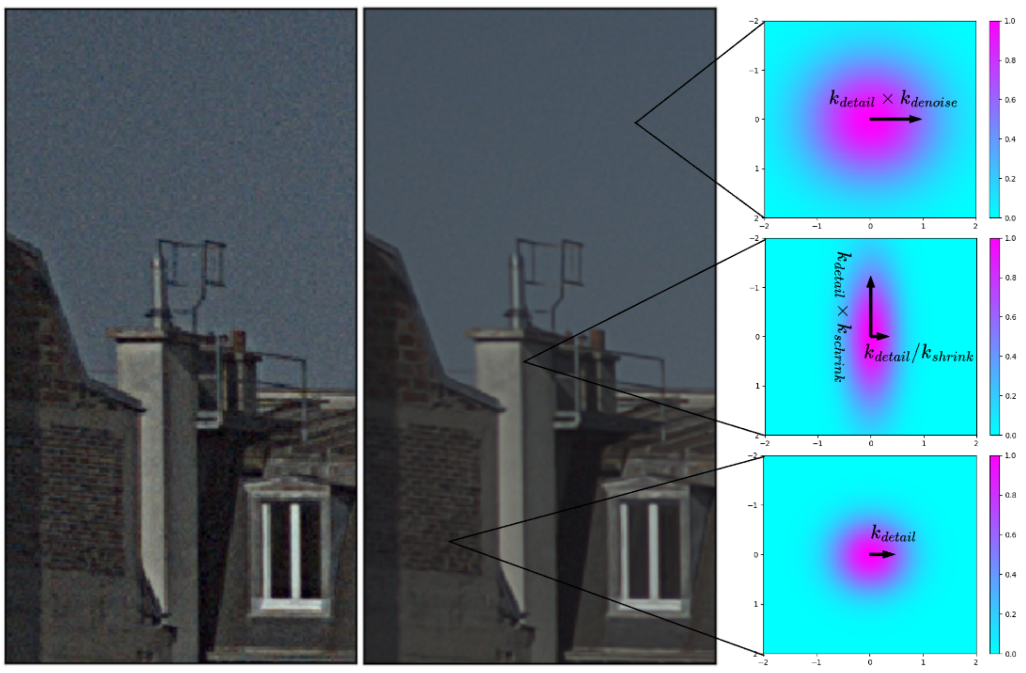

$k_{detail} \times k_{denoise}$
$k_{detail} \times k_{stretch}$
k_{detail}/k_{shrink}
k_{detail}

Figure 5: Illustration of three extreme cases of kernel shape. On the left, the reference image is demosaicked using [20] and upscaled to a factor 2 using nearest neighbors interpolation. On the right, the image upscaled to a factor 2 using the handheld pipeline. Kernels are isotropic with standard deviation $k_{detail} \times k_{denoise}$ in the case of a feature-less local geometry like the sky, isotropic with standard deviation k_{detail} in the case of a punctual detail such as brick contour, or non-isotropically stretched along an edge like the side of a chimney.

- isotropic with std k_{detail} when there is a clean corner ($D \approx 0$ and $A \approx 1$);
- isotropic with std $k_{detail} \times k_{denoise}$ when there is mostly noise ($D \approx 1$ and regardless of A); and
- anisotropic with directional standard deviation values $k_{detail} \times k_{stretch}$ and k_{detail}/k_{shrink} respectively along and across a noise-free edge ($D \approx 0$ and $A \approx 2$).

The different shapes taken by the averaging kernels are illustrated in Figure 5. The original paper defines the directional standard deviation values (k_1, k_2) for the basis directions (e_1, e_2) associated to the eigendecomposition in Equation (7)

$$k_1 = k_{detail}\left((1-D)\frac{1}{Ak_{shrink}} + Dk_{denoise}\right), \tag{12a}$$
$$k_2 = k_{detail}\left((1-D)Ak_{stretch} + Dk_{denoise}\right). \tag{12b}$$

However, it is easy to see that they do not fit the expected behavior described above, especially for the definition of A given in the same article. We thus have to find a substitute strategy to compute correct values of k_1 and k_2.

We remark that the isotropic behavior (where the size of the support varies between k_{detail} and $k_{detail} \times k_{denoise}$) is well captured by the formulas above when we set A, $k_{stretch}$ and k_{shrink} all to 1. Thus, in order to respect the design of the kernel two functions f_1 and f_2 must be found, such that

$$k_1 = k_{detail}\left((1-D)f_1(k_{shrink}, A) + Dk_{denoise}\right), \tag{13a}$$
$$k_2 = k_{detail}\left((1-D)f_2(k_{stretch}, A) + Dk_{denoise}\right). \tag{13b}$$

These functions simply have to model how we want to stretch and shrink the kernel when we are next to an edge, based on A, k_{shrink} and $k_{stretch}$, with the following constraints:

1. $f_1(k_{shrink}, 1) = 1$;
2. $f_1(k_{shrink}, 2) = 1/k_{shrink}$;
3. $f_2(k_{stretch}, 1) = 1$;
4. $f_2(k_{stretch}, 2) = k_{stretch}$.

Simple linear functions such as

$$f_1(k_{shrink}, A) = \frac{1}{k_{shrink}}(A-1) + (2-A), \tag{14a}$$

$$f_2(k_{stretch}, A) = k_{stretch}(A-1) + (2-A), \tag{14b}$$

are valid candidates that are the corrected versions of the linear expressions proposed by Wronski et al. We however posit that non-linear functions, e.g., polynomials or clipping functions, may be better candidates but we leave this tuning to the interested readers. Our implementation managed to provide coherent results using the following threshold functions

$$f_1(k_{shrink}, A) = \begin{cases} \frac{1}{k_{shrink}} & \text{if } A > 1.9, \\ 1 & \text{otherwise;} \end{cases} \tag{15a}$$

$$f_2(k_{stretch}, A) = \begin{cases} k_{stretch} & \text{if } A > 1.9, \\ 1 & \text{otherwise.} \end{cases} \tag{15b}$$

In this setting, we go from full isotropy, when $A = 1$, in the case of sharp detail or complete absence of feature, to exaggerated anisotropy, when $A = 2$ in the case where an edge needs to be dezippered.

Since our registration method is especially efficient in the case of sharp features, relying on a very narrow kernel is not expected to cause artifacts in this case. On the contrary, we have to enlarge the support of the kernel on edges, and we chose to stretch it along the edge to avoid further blurring and zippering. The final matrix Ω in Equation (4) reads

$$\Omega = B^\top \begin{bmatrix} k_1^2 & 0 \\ 0 & k_2^2 \end{bmatrix} B. \tag{16}$$

Lastly, because Ω is computed on the grayscale Bayer images that are at half the resolution of the coarse images J_n ($n = 1, \ldots, N$), we query the values of Ω at the fine image resolution through bilinear interpolation as explained above. An example of such query is shown in Figure 6. In practice, we precompute the matrices Ω at the coarser grids prior to starting the accumulation of J_n within I to save some computations since a local matrix is shared between many pixels of J_n. The pseudo-code for the whole computation of this weights is shown in Algorithm 5.

2.2.2 Robustness Weight

We detail in Algorithm 6 the strategy of [25] to discard the contributions of strongly-misaligned frames or outlier values during fusion. In a nutshell, at each pixel location (x, y) a coefficient R between 0 and 1 is first computed, the coefficient r_n of (3) is later derived from it. The coefficient R measures the ratio between the pixel difference d and the local standard deviation σ and is defined as

$$R(x, y) = \max\left(\min\left(s \times \exp\left(-\frac{d^2}{\sigma^2}\right) - t,\ 1\right),\ 0\right), \tag{17}$$

Algorithm 5: `ComputeKernelCovariance`

```
Data: Image $J$, noise model $(\alpha, \beta)$
Result: Covariance matrix of the kernels at coarse gray positions $\Omega$
$G \leftarrow$ DecimateTogray(GAT($J$,$\alpha$,$\beta$));
$H, W \leftarrow$ Shape($G$);
$G_x \leftarrow$Zeros(H, W-1);
$G_y \leftarrow$Zeros(H-1, W);
$J_x \leftarrow$Zeros(H-1, W-1);
$J_y \leftarrow$Zeros(H-1, W-1);
// Estimating gradients by finite forward differentiation.
for $(i, j) \in [1, H] \times [1, W]$ do
    if $i < H$ then
        $G_y(i, j) \leftarrow (G(i + 1, j) - G(i, j))/2$;
    if $j < W$ then
        $G_x(i, j) \leftarrow (G(i, j + 1) - G(i, j))/2$;

end
// Interpolating gradients on gray pixel corners (cf. text).
for $(i, j) \in [1, H - 1] \times [1, W - 1]$ do
    $J_x(i, j) = (G_x(i + 1, j) + G_x(i, j))/2$;
    $J_y(i, j) = (G_y(i, j + 1) + G_y(i, j))/2$;
end
// One covariance is estimated at the center of each gray pixel
for $(x, y) \in [1, H] \times [1, W]$ do
    $M(0, 0) \leftarrow$ Zeros(2, 2);
    for $p \in \mathcal{N}_2(x, y)$ do
        $M(0, 0) \leftarrow M(0, 0) + J_x^2(p)$;
        $M(0, 1) \leftarrow M(0, 1) + J_x(p)J_y(p)$;
        $M(1, 0) \leftarrow M(1, 0) + J_x(p)J_y(\mathrm{p})$;
        $M(1, 1) \leftarrow M(1, 1) + J_y^2(p)$;
    end
    // Build the covariance matrix
    $\lambda_1, \lambda_2, B \leftarrow$ EigenDecomposition($M$);
    $A \leftarrow 1 + \sqrt{(\lambda_1 - \lambda_2)/(\lambda_1 + \lambda_2)}$;
    $D \leftarrow \max\left(\min\left(1 - \sqrt{\lambda_1}/D_{tr} + D_{th},\ 1\right),\ 0\right)$;
    $k_1 \leftarrow f_1(k_{shrink}, A)$;
    $k_2 \leftarrow f_2(k_{stretch}, A)$;
    $k_1 \leftarrow k_{detail}\left[(1 - D)k_1 + Dk_{denoise}\right]$;
    $k_2 \leftarrow k_{detail}\left[(1 - D)k_2 + Dk_{denoise}\right]$;
    $\Omega(x, y) \leftarrow B^\top \begin{bmatrix} k_1^2 & 0 \\ 0 & k_2^2 \end{bmatrix} B$;
end
```

where the scalars s and t are calibrated beforehand. The pixel difference d and the local standard deviation σ are computed on 3×3 neighborhoods. Wronski et al. claim that this ratio is sufficient to detect if merging a pixel would increase the quality of the final frame, or may introduce visual artifacts. If the accumulation is deemed too "risky", d is expected to be much larger than σ and the

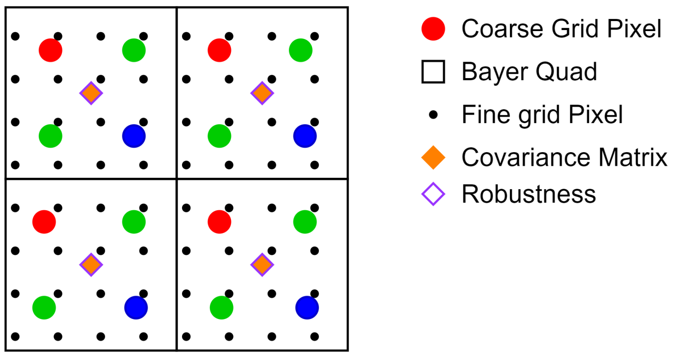


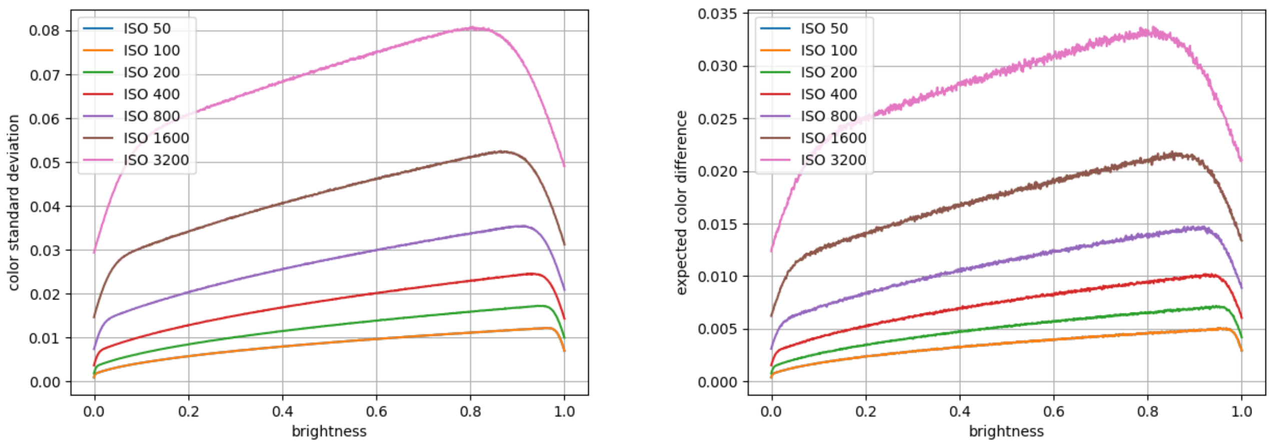
Figure 6: Superposition of the fine grid and the coarse grid of one image of the burst (not necessarily the reference). The covariance matrices and the robustness coefficients are known at the same positions because both are obtained from decimating Bayer quads. For a given queried pixel on the fine grid, the position of the 4 closest covariance matrices are fetched to perform a bilinear interpolation, and the closest robustness coefficient is fetched and used as nearest neighbor interpolation.

pixels will be discarded by enforcing $R = 0$. On the contrary, a d much lower than σ indicates a safe aggregation, enforced by $R = 1$. When pixels are likely aliased, d and σ are comparable. By carefully tuning the parameters s and t, R should allow the accumulation. In what follows, we first compute from the patches only the local mean color difference d_p and standard deviation σ_p, and second, we include the noise model to these measurements via two thresholds d_t and σ_t (whose derivations are detailed in what follows). It ultimately yields the *corrected* values d and σ.

Initial estimates for σ and d. If we handle grayscale images, we simply compute the local mean μ_p and standard deviation σ_p per 3×3 overlapping patches. If instead we have bursts of Bayer images, Wronski et al. build "guide" frames $G_1, \ldots, G_n$ to obtain the local statistics. These half-resolution RGB images are obtained by averaging the two green Bayer green channels, from which we further compute the RGB local statistics on the 3×3 color patches. In that case μ_p is the *color mean* and σ_p is the *color standard deviation*, they are vectors of size 3 computed as the ℓ_2 norm of the vector whose components are the standard deviation of each color channel. By using the patchwise flows between J_1 and J_n we find the corresponding local means on the two images and compute the ℓ_2 norm of the difference to predict the *local color difference* d_p.

Leveraging the noise model. A typical deghosting algorithm would simply predict a weight based on d_p, yet Wronski et al. go a step further and introduce the noise model [9] to correct the measurement d_p that may be too large due to noise and not misalignment. Indeed, d_p and σ_p are computed using too few samples to neglect the effect of noise. To compensate this effect, the original paper introduces prior information using the noise model. The noise is expected to follow the Poissonian-Gaussian model, commonly approximated as an heteroscedastic Gaussian noise with a variance evolving linearly with brightness [9]. This model is used to determine, for any given brightness level, the expected color variance σ_t and the expected color difference d_t, both also of size 3, between two identical patches while considering the effect of noise.

Figure 7 shows an example of such curves. At a given pixel location, we query the local average brightness level for each channel μ_p, the corresponding values σ_t and d_t from the curves, and correct the empirical values of σ_p and d_p via two separate thresholding operators (note that these operations



(a) Expected standard deviation σ_t. (b) Expected color difference d_t.

Figure 7: Experimentally generated noise model for the Google Pixel 4a camera. Because of the sensor clipping around 0 and 1, the expected standard deviation (Figure 7a) is lower than when predicted using the heteroscedastic Gaussian model.

are with vector quantities) for all color channel c in $\{R, G, B\}$

$$\sigma_c = \max(\sigma_{p,c}, \sigma_{t,c}), \tag{18a}$$

$$d_c = \frac{d_{p,c}^2}{d_{p,c}^2 + d_{t,c}^2} d_{p,c}. \tag{18b}$$

Lastly, we compute the scalar quantities d and σ featured in Equation (17) by taking the norms

$$d = \left(\sum_{c \in \{R,G,B\}} d_c^2 \right)^{1/2} \quad \text{and} \quad \sigma = \left(\sum_{c \in \{R,G,B\}} \sigma_c^2 \right)^{1/2}. \tag{19}$$

Note that $\sigma_{t,c}$ and $d_{t,c}$ are implicitly defined as functions of $\mu_{p,c}$: $\sigma_{t,c} = \sigma_{t,c}(\mu_{p,c})$ and $d_{t,c} = d_{t,c}(\mu_{p,c})$ where by abuse of notations we confound the curves $\mu_{p,c} \mapsto \sigma_{t,c}(\mu_{p,c})$ and $\mu_{p,c} \mapsto d_{t,c}(\mu_{p,c})$ with the queried values $\sigma_{t,c}$ and $d_{t,c}$. The first equation applies hard thresholding by below the value of σ_p. This is because in the vanilla model, lower values of predicted noise variance would lead to very large ratio values, and thus robustness of 0. As a result few frames are used, thus favoring zipper and noise in the prediction. Since visually it is better to have a slightly blurry but noise-free image than a noisy sharp one, clipping by below results in more samples being retained to indeed better denoise at the cost marginal additional blur. The second equation is a Wiener shrinkage applied to $d_{p,c}$ that undermines the impact of the noise that necessarily increases the mean color difference between to constant patches, resulting in lower confidence values. Note that both operation make sure that the corrected ratio d/σ is always smaller than the one featuring d_p and σ_p, forcing R to be closer to 1 compared to the initial estimates, and thus being more likely to accumulate samples.

Upscaling and warping the local statistics. An important problem is to upscale and warp the local statistics σ_p and μ_p to the resolution and pose of the LR reference frame since they are computed on the guided images G_n that are twice smaller. Note that we must have the μ_p's in the same pose to compute the difference vectors d, further justifying the need to interpolate the local statistics. We could virtually use any interpolator such nearest neighbor to go fast, but it may be prone to aliasing on detailed areas, thus ultimately resulting in more rejection than wanted. It is thus crucial to resample the statistics with an efficient and fast anti-aliasing kernel. The paper does not mention any strategy for doing so but the 3×3 quadratic approximation of the cubic filter proposed by Dodgson [5] is a good candidate since it is fast, does not introduce sub-pixel shift since its support is odd, and is robust to aliasing. This stage yields robustness maps r_n ($n = 1, \ldots, N$) of size $H \times W$, like the LR frames.

Penalizing the stronger motions. After correction with the noise model, we rescale and add an intercept to $\exp(-d^2/\sigma^2)$ with the hyper-parameters t and s. The former is fixed after calibration to 0.12 whereas the latter is a patch-wise quantity that depends only on the magnitude of the motion field and takes values of either $s_1 = 2$ or $s_2 = 12$. We now detail how to choose between s_1 and s_2 based on the optical flow measurements. Recall that we denote by $V_n^{(p)}$ the local translation for the p-th non-overlapping patch ($p = 1, \ldots, P$) in the n-th frame ($n = 1, \ldots, N$). By calling V_x and V_y the two translations of $V_n^{(p)}$ for a given pair (p, n), the decision to select s among s_1 and s_2 reads

$$M_x = \max_{p' \in \mathcal{N}_p^3} V_x(p') - \min_{p' \in \mathcal{N}_p^3} V_x(p'), \tag{20a}$$

$$M_y = \max_{p' \in \mathcal{N}_p^3} V_y(p') - \min_{p' \in \mathcal{N}_p^3} V_y(p'), \tag{20b}$$

$$M = \sqrt{M_x^2 + M_y^2}, \tag{20c}$$

$$s(p) = \begin{cases} s_1 \text{ if } M > M_t, \\ s_2 \text{ otherwise,} \end{cases} \tag{20d}$$

where $\mathcal{N}_p^3$ is a 3×3 patch neighborhood around p. Put in words, it means that for each patch location p, we find the largest and smallest patchwise flows in the local neighborhood to measure the local motion variation M. The rationale is that the larger the motion, the less reliable is the patch we want to aggregate and it may lead to unacceptable visual artifacts. Thus, we assign a smaller coefficient value s_1 if the motion variation is larger than the threshold M_t, otherwise we assign to the patch a larger coefficient s_2 that scales the local robustness weight R. Using $s_1 > 0$ still enables to aggregate patches whose textures are perfect matches, despite a strong alignment variation. This is typical of sky crops for instance, where the estimated alignment may feature large local motion because the alignment task is ill-posed, as mentioned earlier. Setting $s_1 = 2$, six times smaller than s_2 for the favorable smooth motion case is a good compromise found in practice in [25].

Having selected s between s_1 and s_2, we have fully determined how to compute the local robustness R in (x, y). In [25], a final step introduces more spatial coherence among the locally-computed robustness coefficients solely by selecting the minimum of R in a 5×5 neighborhood centered in each pixel location (x, y)

$$r(x, y) = \min_{(u,v) \in \mathcal{N}^5} R(u, v). \tag{21}$$

This final step favors the most pessimistic confidence scores to further prevent visual artifacts. Algorithm 6 shows the pseudo-code for computing the robustness weight r, and Algorithms 7, 8 and 9 show the companion routines for computing the guide images, the local statistics, and local minima.

Algorithm 6: `ComputeRobustness`

```
Data: Reference image $J_1$, $n$-th moving image $J_n$, optical flow reference/$n$-th frame $V_n$, noise
      curve $\mu_p \mapsto \sigma_t$, pixel difference curve $\mu_p \mapsto d_t$, motion threshold $M_t$, spatial coefficients
      $(s_1, s_2)$, coefficient bias $t$
Result: Robustness map $r$
// Initialization
1  $H, W \leftarrow$ Shape$(J_1)$;
2  $r \leftarrow$ Zeros$(H, W)$;
3  $R \leftarrow$ Zeros$(H, W)$;
4  $\mu_p \leftarrow$ Zeros$(H/2, W/2, 3)$;
5  $\hat{\mu}_p \leftarrow$ Zeros$(H/2, W/2, 3)$;
6  $\sigma_p \leftarrow$ Zeros$(H/2, W/2, 3)$;
7  $\hat{\sigma}_p \leftarrow$ Zeros$(H/2, W/2, 3)$;
8  $G_1 \leftarrow$ ComputeGuideImage$(J_1)$;
9  $G_n \leftarrow$ ComputeGuideImage$(J_n)$;
10 $s \leftarrow$ ZerosLike$(V_n)$;
// Initial estimate
11 for $(x, y) \in G_1$ do
       // Local statistics computation
12     $\widehat{\mu}_p(x, y), \widehat{\sigma}_p(x, y) \leftarrow$ ComputeLocalStatistics$(G_1, (x, y), 3)$;
13     $\mu_p(x, y), \sigma_p(x, y) \leftarrow$ ComputeLocalStatistics$(G_n, (x, y), 3)$;
14 end
// We iterate on patches p in the image support to update the local value of s.
15 for $p \in s$ do
       // Computing threshold based on flow irregularities
16     $M_x \leftarrow$ ComputeLocalMax$(V_{nx}, p, 3) -$ ComputeLocalMin$(V_{nx}, p, 3)$;
17     $M_y \leftarrow$ ComputeLocalMax$(V_{ny}, p, 3) -$ ComputeLocalMin$(V_{ny}, p, 3)$;
18     if $\sqrt{M_x^2 + M_y^2} > M_t$ then
19         $s(p) \leftarrow s_1$;
20     else
21         $s(p) \leftarrow s_2$;
22     end
23 end
24 for $(x, y) \in R$ do
       // Dodgson $\times 2$ upscaling and warping of the local statistics
25     $\sigma \leftarrow$ DodgsonApproximateQuadratic$(\sigma, 2, V_n)$;
26     $\hat{\mu} \leftarrow$ DodgsonApproximateQuadratic$(\hat{\mu}, 2, V_n)$;
       // Color distance
27     $d_p \leftarrow |\mu_p - \widehat{\mu}_p|$;
       // Noise correction
28     $\sigma \leftarrow \| \max(\sigma_t(\mu_p), \sigma_p) \|_2$;
29     $d \leftarrow \| \left[ d_p^2 / (d_p^2 + d_t(\mu_p)^2) \right] d_p \|_2$;
       // Local robustness weight computation in $[0, 1]$.
30     $p \leftarrow$ patchId$(x, y)$;
31     $R(x, y) \leftarrow \min(\max(s(p) \times \exp(-d^2/\sigma^2) - t, 0), 1)$;
32 end
// Final refinement: neighbor-aware smoothing
33 for $(x, y) \in r$ do
34     $r(x, y) \leftarrow$ ComputeLocalMin$(R, (x, y), 5)$;
35 end
```

Algorithm 7: `ComputeGuideImage`

```
Data: Raw image $I$
Result: Guide image $G$
$H, W \leftarrow$ Shape($I$);
$G \leftarrow$ Zeros($H/2, W/2, 3$);
for $(x, y) \in G$ do
    $G(x, y, 0) \leftarrow I(2x, 2y)$;
    $G(x, y, 1) \leftarrow (I(2x, 2y+1) + I(2x+1, 2y))/2$;
    $G(x, y, 2) \leftarrow I(2x+1, 2y+1)$;
end
```

Algorithm 8: `ComputeLocalStatistics`

```
Data: Guide image $G$, pixel location $(x, y)$, window size $S$
Result: Local RGB mean $\mu$, local RGB standard deviation $\sigma$
$\mu \leftarrow$ Zeros($3$);
$\sigma^2_{RGB} \leftarrow$ Zeros($3$);
// Compute local statistics per color channel
for $c \in \{0, 1, 2\}$ do
    for $(u, v) \in \mathcal{N}_S(x, y)$ do
        $\mu(c) \leftarrow \mu(c) + G(u, v, c)$;
        $\sigma^2_{RGB}(c) \leftarrow \sigma^2_{RGB}(c) + G(u, v, c)^2$;
    end
end
$\mu \leftarrow \mu/S^2$;
$\sigma^2_{RGB} \leftarrow \sigma^2_{RGB}/S^2 - \mu^2$;
// We aggregate the three color variances into a single scalar as our final estimate.
$\sigma \leftarrow \sqrt{\sigma^2_{RGB}(0) + \sigma^2_{RGB}(1) + \sigma^2_{RGB}(2)}$;
```

Algorithm 9: `ComputeLocalMin`

```
Data: Image $I$, pixel location $(x, y)$, window size $S$
Result: Local minimum $r$
$r \leftarrow R(x, y)$;
for $(u, v) \in \mathcal{N}_S(x, y) - \{(x, y)\}$ do
    if $r > R(u, v)$ then
        $r \leftarrow R(u, v)$;
    end
end
```

2.3 Additional Implementation Details

Besides the main structure of the algorithm that is composed of the registration and fusion main modules, there are additional implementation details to render better images that are barely evoked in [25]. We also strove to reproduce them in this article.

Computing robustness noise model parameters. An important part of the robustness module is including the noise model into the local measurements of the standard deviation and the color differences. The original paper [25] refers to the calibration of the parameters σ_t and d_t as the result of "a series of Monte Carlo simulations for different brightness levels" that we have reimplemented. Algorithm 10 details each stage of this simulation.

Algorithm 10: `Monte Carlo simulation`

```
Data: Brightness intervals $B$, number of trials $N$, noise model $(\alpha, \beta)$, ISO gain $g$
Result: Expected per-brightness standard deviation $\widehat{\sigma}$ and mean difference $\widehat{d}$
1  $\widehat{\sigma} \leftarrow \texttt{zeros}(B+1)$;
2  $\widehat{d} \leftarrow \texttt{zeros}(B+1)$;
3  for $b = 0$; $b \leq B$; $b \leftarrow b+1$ do
4      for $n = 1$; $n \leq N$; $n \leftarrow n+1$ do
           // Simulate two constant $3 \times 3$ patches at brightness $b/B$ in $\{0, 1/B, \dots, 1\}$
5          $I_1 \leftarrow b/B \times \texttt{ones}(3,3)$;
6          $I_2 \leftarrow b/B \times \texttt{ones}(3,3)$;
           // Add noise and clip
7          $I_1 \leftarrow \texttt{clip}(I_1 + g\sqrt{\alpha I_1/g + \beta} \times \texttt{randn}(I_1.\texttt{shape}), 0, 1)$;
8          $I_2 \leftarrow \texttt{clip}(I_2 + g\sqrt{\alpha I_2/g + \beta} \times \texttt{randn}(I_2.\texttt{shape}), 0, 1)$;
           // Compute the local statistics
9          $\sigma \leftarrow (\texttt{std}(I_1) + \texttt{std}(I_2))/2$;
10         $\mu_1 \leftarrow \texttt{mean}(I_1)$;
11         $\mu_2 \leftarrow \texttt{mean}(I_2)$;
12         $d \leftarrow |\mu_1 - \mu_2|$;
           // Add these estimates to the buffers
13         $\widehat{\sigma}(b) \leftarrow \widehat{\sigma}(b) + \sigma/N$;
14         $\widehat{d}(b) \leftarrow \widehat{d}(b) + d/N$;
15     end
16 end
```

We generate uniform 3×3 patches at a constant brightness in an interval $\{0, 1/L, \dots, 1\}$ with L a certain interval value, *e.g,* 1,000. For each patch, we generate from a given noise curve, i.e., the Poissonian and Gaussian noise parameters proper to a camera sensor, two instances noises that we apply to the patches, followed by clipping between 0 and 1 to simulate the sensor non-linearities. From these two patches we compute the color standard deviation and the color mean. We add the average of the standard deviation to the buffer of expected value of σ_t, and compute the Euclidean distance between the color means of the two patches and add it to the buffer of the expected value of d_t. After an important amount of trials, e.g., 10,000 trials, we have the Monte Carlo expected values for σ_t and d_t. We do so for each brightness value in $\{0, 1/L, \dots, 1\}$ to have a dense estimate of the noise model over the whole range of values, including the non-linearities next to the black and white points. An example of the curves can be seen in Figure 7.

Noise level-aware parameters. The size k_{detail} and $k_{denoise}$, as well as the thresholds D_{th} and D_{tr} in the computation of D, and the patch size T, are all functions of an estimated signal-to-noise ratio (SNR). The rationale behind this design is to automatically tune these parameters to better handle the adversarial effect of noise for alignment and detection of salient edges "based on perceptual image quality assessment ensuring visual consistency for SNR values from 6 to 30 where the SNR was measured from a single frame", to quote Wronski et al. [25]. For instance, for stronger noise levels the details in tiny patches may not be easily told apart from noise, whereas larger patches increase the chances of finding a salient-enough edge for safely computing the optical flow. Two questions arise for implementing this features: How do we compute the SNR, and what are the functions of the SNR used to select a relevant value for each of these parameters?

In [25], the SNR is "computed based on the average image brightness and the noise model". We have observed that in practice using the noise model to compute

$$\mathrm{SNR} = \frac{b}{\sigma_t(b)} \qquad \text{with} \tag{22}$$

$$b = \frac{1}{HW} \sum_{x,y \in J_1} J_1(x,y) \tag{23}$$

is a reliable and computationally efficient estimation of the SNR. Evaluating the noise curve σ_t at b means that we query the clipped Poissonian-Gaussian model of the expected noise standard deviation obtained with the Monte-Carlo simulator at the average brightness level in the image. Hyper-parameters such as $k_{detail}, k_{denoise}, D_{th}$ and D_{tr} are set to evolve linearly with the SNR *clipped* in the range $[6, 30]$. For instance for k_{detail}, this model is implemented as follows

$$k_{detail} = (k_+ - k_-) \times (\mathrm{SNR} - 6)/(30 - 6) + k_-, \tag{24}$$

where k_- and k_+ are predetermined extreme values correspond respectively to the lowest and highest SNR. The resulting SNR-specific values for the lowest (resp. highest) SNR score are 0.33px (resp. 0.25px) for k_{detail}, 5.0 (resp. 3.0) for $k_{denoise}$, 0.81 (resp. 0.71) for D_{th}, and 1.24 (resp. 1) for D_{tr}. T is 64 if the SNR is less than 14, 32 if the SNR is between 14 and 22, or 16 if the SNR is above 22.

Noise-agnostic parameters. In addition to T, k_{detail}, $k_{denoise}$, D_{th}, and D_{tr}, the following parameters are noise-agnostic: $k_{stretch}$ is 4, k_{shrink} is 2, t is 0.12, s_1 is 2, s_2 is 12 and M_{th} is 0.8px.

Compensating for the lack of accumulated frames. In many situations, such as for occlusions, the robustness retains only a few frames from the stack of N images to compute the local HR entry $I(x, y)$. To compensate the lack of samples that may lead to blatant artifacts and weaken temporal denoising, Wronksi et al. mention an "additional spatial denoising with strength inversely proportional to the merged frame count" to compensate for the aforementioned artifacts.

Automatically tuning a denoising threshold according to the number of frames to be merged was already proposed by the same research team in [16] in the context of low-light photography. We may reasonably posit that in [25] the authors implemented that solution to adapt spatial denoising to the number of frames actually used during fusion (the ones not rejected by robustness). However it is not crystal clear what they did in practice, and we instead came up with a substitute post-processing strategy that also boils down to shaping a spatial denoising kernel inversely proportional to the value of the accumulated robustness, i.e., an estimate of the number of frames fused.

The robustness maps r_n $(n = 1, \ldots, N)$ are designed to quickly vary between 0 and 1. We use them to build the accumulated robustness map defined for a pixel location (x, y) as

$$R_c(x,y) = \sum_{n=2}^{N} r_n(x,y), \tag{25}$$

as a proxy for the number of frames actually retained for collaborating at this given location for interpolating the high-resolution image. Note that we do not take into account the reference frame since $r_1(x, y) = 1$ everywhere. During the sequential aggregation for computing the merge in Equation (3), we thus start by accumulating the frames $n = 2, \dots, N$ and compute the accumulated robustness on the fly. When ultimately adding the reference frame as in Equation (3), we may expand the vanilla 3×3 support of the merging kernel Ω for the locations (x, y) where $R_c(x, y)$ is smaller than an arbitrary threshold R_{max} controlling if enough frames are locally contributing. If R_c is larger than R_{max}, we compute the covariance matrix Ω as before. Otherwise, we may lack of samples, which leads to artifacts. In that case, we need to collect more neighboring pixels with a wider covariance matrix Ω'. Since Ω' is designed to be larger than Ω, the integration support $\mathcal{N}$ should also be increased to allow a wider aggregation than the original 3×3 window. This can be summarized as

$$\Omega' = g_1(R_c) \times \Omega, \tag{26a}$$
$$\mathcal{N} = g_2(R_c), \tag{26b}$$

where we handcraft the constraints on g_1 and g_2,

1. $g_1(R_c > R_{max}) = 1$;
2. $g_1(R_c \leq R_{max}) \geq 1$;
3. $g_2(R_c > R_{max}) = \mathcal{N}_3$;
4. $\mathcal{N}_3 \subset g_2(R_c \leq R_{max})$.

We consider the simple yet illustrative example

$$g_1(R_c) = \begin{cases} 1 & \text{if } R_c > R_{max}, \\ C \text{ with } C > 1 & \text{otherwise.} \end{cases}, \tag{27a}$$
$$g_2(R_c) = \begin{cases} \mathcal{N}_3 & \text{if } R_c > R_{max}, \\ \mathcal{N}_5 & \text{otherwise.} \end{cases}. \tag{27b}$$

In the experiment section we will see that the empirical values $C = 8$ and $R_{max} = 8$ provide satisfactory results. A better filter may be obtained by tuning C to evolve smoothly with R_c.

Moreover, we have remarked that areas where R_c is low are generally prone to sharp fusion artifacts, *e.g.,* non-rigid moving objects or persons, despite the prior application of the robustness mask. In practice, low R_c values correspond to retaining a couple frames over more than ten or twenty, which is not enough to achieve correct image fusion. To avoid such scenarios, we switch from a multi-frame SR approach to a single-frame upscaling strategy in these areas by completely discarding values accumulated so far, and relying solely on the reference frame. This prevents blatant artifacts in the case of important displacements and deformations across frames, for instance. Other fusion strategies based on the value of R_c may also be as correct since the details on this module are barely evoked in [25, Section 7.4].

This alternative accumulation logic is detailed in Algorithm 11, which is featured at the end of the main Algorithm 1. Note that one may use the same accumulation algorithm (Algorithm 4) as for the other frames, but with the risk of unacceptable visual artifacts in the final HR image.

We have also compared this method with an alternate post-processing strategy applying a Gaussian or median filter on the final HR image I obtained from Equation (3), whose support is inversely proportional to R_c. Not only this second approach is less computationally efficient, but we have also observed that the resulting image appears more blurry and retains more fusion artifacts.

Algorithm 11: `AccumulationReference`

```
Data: Frame $J_1$, kernels $\Omega_1$, zoom $s$, numerator image num, denominator image den,
      accumulated robustness $R_c$
Result: Updated numerator and denominator of the accumulator: num and den
// Main loop over the pixels of the fine grid
1  for $(x, y) \in num$ do
       // Corresponding location on $J_1$'s coarse grid
2      $(x_1, y_1) \leftarrow (x, y)/s$;
       // Fetch the 4 closest pre-computed covariance matrices
3      for $(\hat{x}_1^i, \hat{y}_1^i) \in \mathcal{N}_2(x_1, y_1)$ do
4          $\Omega^i \leftarrow$ Fetch$(\Omega_1, (\hat{x}_1^i, \hat{y}_1^i))$;
5      end
       // Interpolate to the desired position based on the known Omegas
6      $\Omega \leftarrow$ BilinearInterpolation$(\Omega^1, \Omega^2, \Omega^3, \Omega^4, x_1, y_1)$;
7      $r_c \leftarrow$ NearestInterpolation$(R_c, (x_1, y_1))$;
       // Expanding Omega's support and the integration neighborhood
8      $\Omega \leftarrow g_1(r_c) \times \Omega$;
9      $\mathcal{N} \leftarrow g_2(r_c)$;
10     if $r_c < R_{max}$ then
           // Discarding the aggregation prone to artifacts to switch to a single-frame strategy
11         num$(x, y) \leftarrow$ Zeros$(3)$;
12         den$(x, y) \leftarrow$ Zeros$(3)$;
13
14     for $(u, v) \in \mathcal{N}(x_1, y_1)$ do
           // Build on-the-fly the merge kernel with $(u, v)$ back-projection on $I$'s fine grid
15         $d \leftarrow (u, v) - (x_1, y_1)$;
16         $w \leftarrow \exp\left(-d^\top \Omega^{-1} d/2\right)$;
           // Look which color component will be added
17         $c \leftarrow$ GetLocalChannel$((u, v))$;
           // Do the aggregation in the image and accumulator
18         num$(x, y, c) \leftarrow$ num$(x, y, c) + rwJ_1(u, v)$;
19         den$(x, y, c) \leftarrow$ den$(x, y, c) + rw$;
20     end
21 end
```

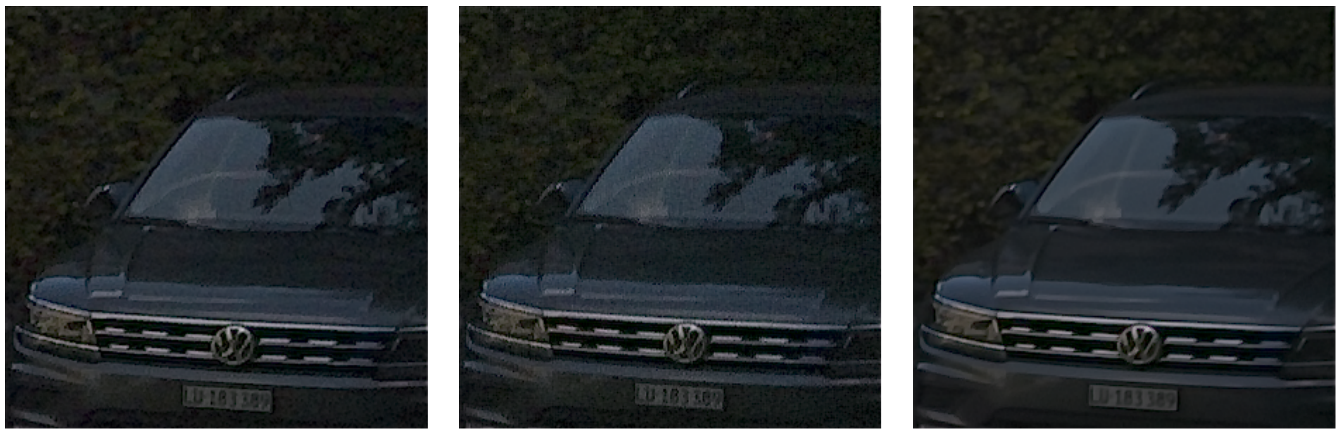
Postprocessing in the demo. In the companion demo shipped with this paper, we have to postprocess the fused image to display it side-by-side with the reference LR frame demosaicked with [19] and upsampled (if s is not set to 1) with bilinear interpolation. Starting from RGB images in the camera's color space, we use gamma compression with value of 2.2, the pixelwise S-curve $x \mapsto 3x^2 - 2x^3$ in guise of tone mapping, and lastly use the implementation of Polyblur from [7] to sharpen the image. The images are saved as 8-bit JPEG files with compression quality set to 95%.

3 Experiments

This section presents experiments validating our implementation, highlighting the demosaicking and super-resolution abilities of the handheld super-resolution algorithm. We process bursts of raw photographs after setting the black and white points, and correcting the white balance with the coefficients specified by the EXIF metadata. The technique replaces the subsequent denoising and demosaicking modules in a typical ISP pipeline. After having merged the frames into a single image, we convert the colors to the sRGB format by applying the color matrix found in the EXIF metadata, sharpen the image and remove chromatic aberrations with the state-of-the-art approach of [8]. Lastly, we apply a 1/2.2 gamma compression curve as a simple global tone mapping operator. Final operations such as 8 or 10-bit quantization and dithering are not considered. In the high-resolution version of this paper, the illustrations of this section are taken from 16-bit TIFF uncompressed images.

3.1 Demosaicking

Since the handheld burst super-resolution algorithm is a drop-in replacement of the demosaicking block in the ISP pipeline, we first measure its ability to compete with the classical handcrafted interpolation algorithm of Menon et al. [20], and the deep learning approach of Gharbi et al. [10]. Demosaicking corresponds to set the zoom factor s to 1. The result in Figure 8 shows that our pipeline is able to demosaick and denoise raw images while conserving sharp details.



(a) Menon et al. [20]. (b) Gharbi et al. [10]. (c) Ours ($N = 13$).

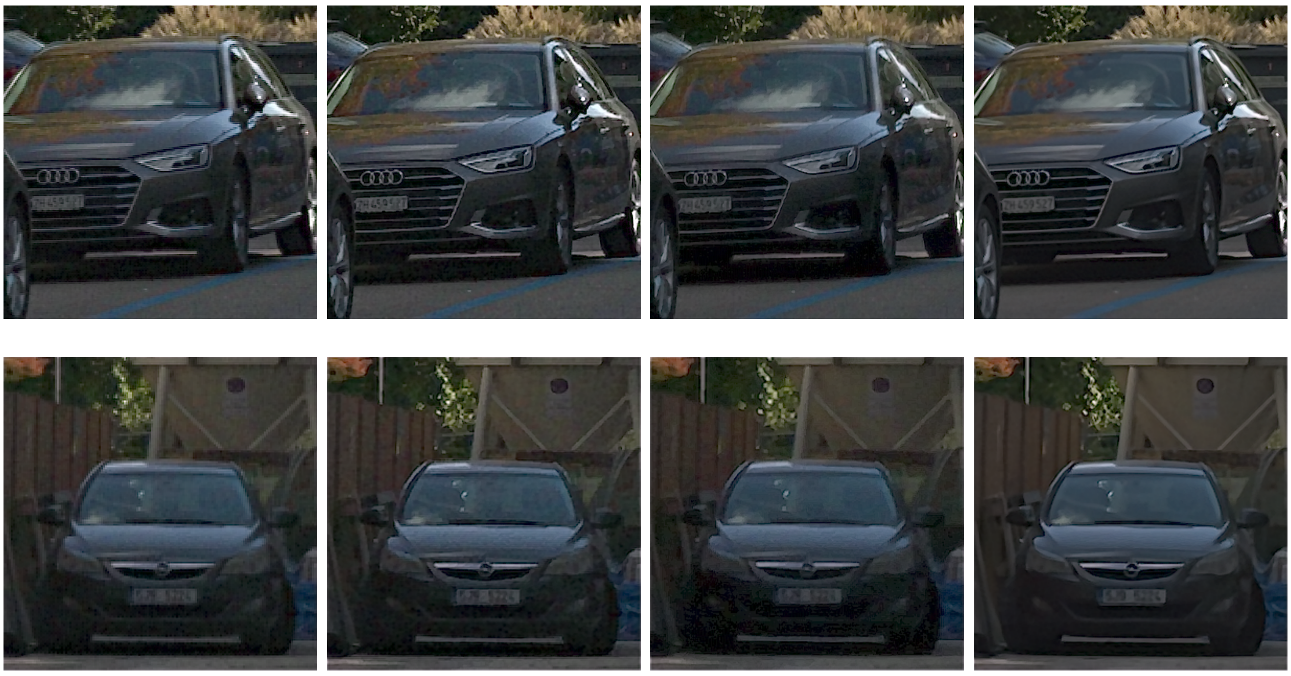
Figure 8: Quantitative demosaicking comparison. Our methods generates a sharp and noise free image. The licence plate is better demosaicked in our case, revealing all the numbers.

Using N images instead of a single one also prevents hallucinated details. The proposed algorithm, when $s = 1$, is thus a drop-in replacement of any demosaicking approach in a typical ISP pipeline, but at the cost of taking several photographs instead of a single one. This is particularly adapted to smartphones that have fully electronic shutters that do not break after a certain amount of clicks.

3.2 Super-resolution

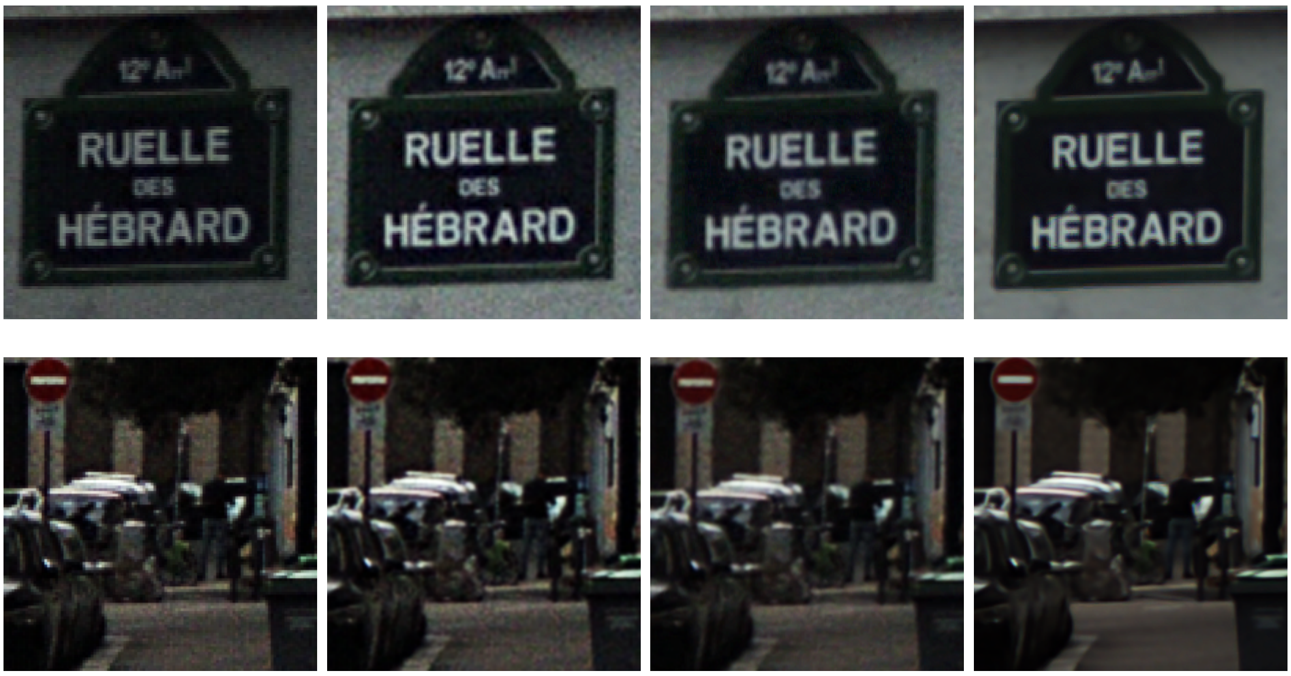
We now evaluate super-resolution with a factor $s = 2$. We compare our approach to the classical bicubic interpolation and the recent deep learning-based video SR approach of [11], which predicts in a recursive manner high-resolution frames from a RGB sequence. Note that [11] is a neural network with a large memory footprint, preventing the restoration of large images on a single GPU. In contrast, our method can process a burst of 12 megapixels (about 4000×3000 pixels) on a single GPU in 3 seconds for zooming with a factor 2, making it far more amenable for real-world applications. Since these two baseline methods only operate on RGB frames, before applying them we demosaick the input frames with demosaicnet [10].

Figures 9 and 10 are images from the bursts of $N = 20$ raw images in [15] and $N = 13$ raw images in [3]. Our method successfully manages to separate details from noise, and the resulting image appears both sharp and denoised.



(a) Reference. (b) Bicubic. (c) Haris et al. [11]. (d) Ours ($N = 13$).

Figure 9: Comparison of different methods for $\times 2$ zoom on a real burst from [3]. The images are demosaicked using [10], upscaled using several methods, then postprocessed using [8]. Our pipeline manages to recover sharp details such as registration plates, while denoising even the darkest areas. Better seen by zooming on a computer screen.



(a) Reference. (b) Bicubic. (c) Haris et al. [11]. (d) Ours ($N = 20$).

Figure 10: Comparison of different methods for $\times 2$ zoom on a real burst from [15]. The images are demosaicked using [10] and upscaled using several methods. Our pipeline manages to accurately recover the shape of the letters while removing the image noise. Better seen by zooming on a computer screen.

3.3 Computation Details

To highlight the efficiency of the algorithm, we share in this section the run time and peak memory usage of our early “naive” implementation. Our Python implementation uses Numba’s just in time compilation (JIT), and relies on PyTorch’s convolution and FFT functions. Our BM module does not use the fast ℓ_2 residual computation trick mentioned in [13]; we represent floating numbers with 32 bits, and do not use fast math approximation for operators such square roots or exponentials. The memory deallocation is also naively left in the hands of the interpreter.

The complexity of the algorithm can be split in two parts. During the initialization, the reference frame is processed, by computing the Hessian matrices and the gradients necessary for the ICA algorithm. This represents a fixed temporal and memory cost.

During the second part, all the remaining frames are processed sequentially: the memory cost is therefore fixed and the temporal cost evolves linearly with the burst size.

We therefore split the following measures into a fixed cost associated with the reference frame, and an additional cost associated to each *supplementary* frame of the burst. We have run all the experiments on a workstation with a NVIDIA RTX 3090 GPU with 24GB of RAM. The demosaicking of a raw burst of $N = 20$ 12Mp photographs is processed in 300ms, with an additional 110ms for each non reference frame and the GPU memory reaches a maximum of 3.8GB. The processing of a $\times 2$ SR for a raw burst of $N = 20$ 12Mp photographs takes 500ms and an additional 190ms per non reference frame, for a peak GPU memory usage of 4.7GB.

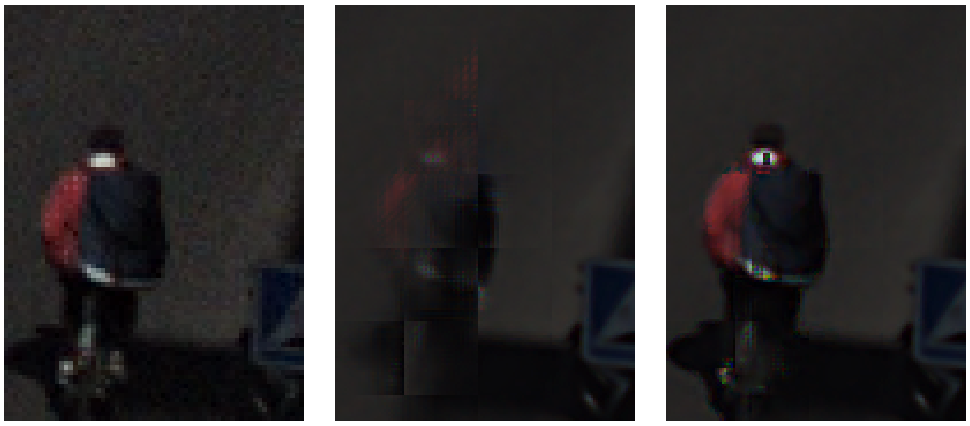
We are aware that our running times on a NVIDIA RTX 3090 GPU are slower than that in [25] for a smartphone Snapdragon processor. Our goal is not to reproduce the performance but to provide an easy-to-read code useful to reproduce the visual results. To accelerate the code, one should implement Algorithm 1 with a lower-level optimized language such as Vulkan, OpenCL, or pure CUDA to avoid the intermediate calls to the Python interpreter. Yet, even without a fully optimized implementation, our pipeline runs considerably faster than state-of-the-art approaches such as [11] or [10]. Note that the memory usage of [11] is so high that processing a 12MP image is impossible, even for the higher-end consumer grade GPUs.

3.4 Impact of the Robustness Term

Wronski et al. [25] run many ablation studies to evaluate the impact of the numerous parameters. In this paper, we focus on the role of the robustness weight R since it is one of the most important components of the pipeline that is a pure innovation of the original paper. Figure 11 shows an example result obtained without robustness and with it.

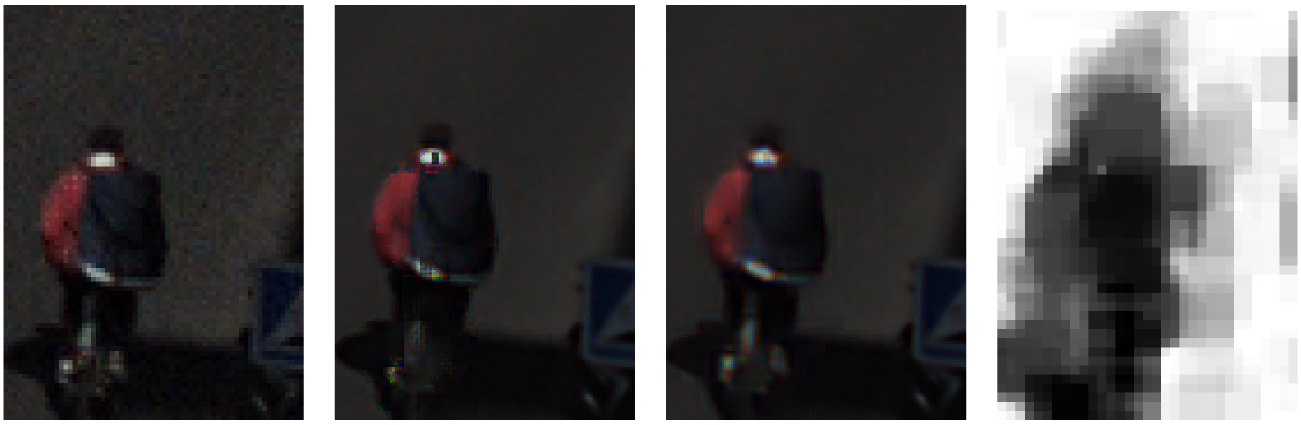
When the robustness is deactivated, we observe unacceptable artifacts caused by the non-overlapping model of the optical flow. Activating the robustness, as described in Section 2.2, removes most artifacts. However, in this context we merge a varying number of pixels per location as shown in the aggregation map (Figure 12d). Black tiles correspond to areas where SR was performed using contributions from a single frame, i.e., where there is motion, and white tiles correspond to areas where the $N = 20$ images are used for fusion, i.e., where the scene is static. Note that areas with a single-frame contribution are prone to demosaicking artifacts.

To prevent this issue we follow [25] and apply a frame-count aware filter to the image. We locally enlarge the merging kernel of the reference frame when the frame count is low following the protocol of Section 2.3. Figure 12c shows the outcome of this strategy. The moving cyclist appears slightly blurrier, but it is not that important for a moving element of the scene, but the unacceptable fusion artifacts are less pronounced, which is a more important criterion for the customers.



(a) Reference. (b) Robustness off. (c) Robustness on.

Figure 11: Example of the robustness for a $\times 2$ zoom of a real burst from [15]. The scene exhibits a moving cyclist moving forward throughout the sequence. Without robustness, the output of the pipeline suffers from heavy ghosting artifacts on moving objects (Figure 11b). Using the robustness weights to reject ill-aligned frames removes this type of artifacts (Figure 11c). Note that static objects such as road signs are left unchanged by the robustness.

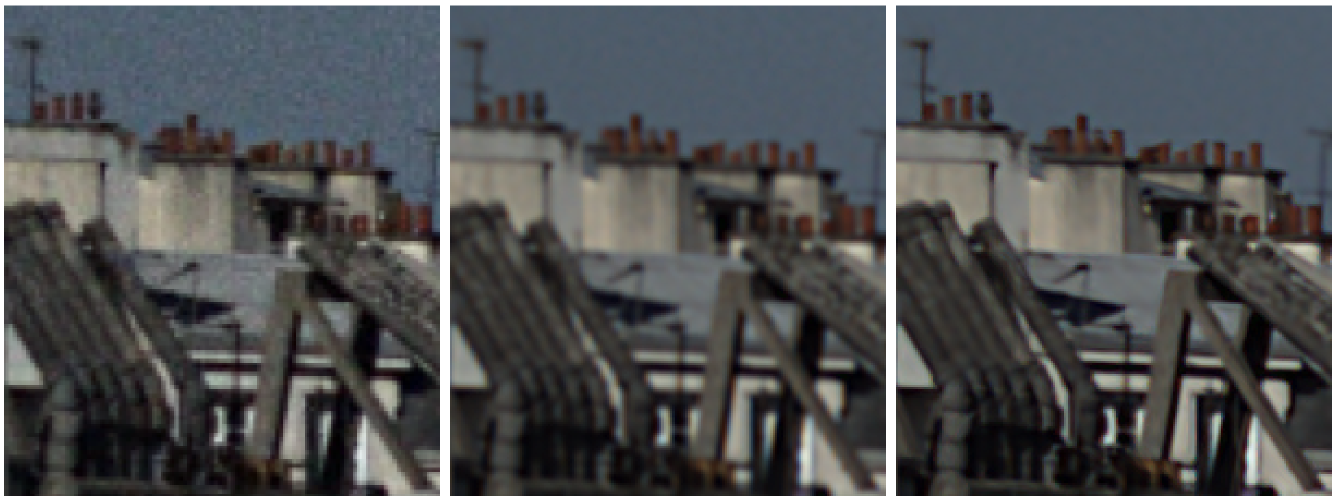


(a) Reference. (b) Robustness on. (c) Corrected robustness on. (d) Accumulated robustness.

Figure 12: Illustration of the additional frame-count aware spatial denoising for a $\times 2$ zoom of a real 20 images burst. With the mere robustness logic, patches with a low accumulated robustness (black pixels in Figure 12d) are mostly obtained using single-frame SR and are therefore exceptionally prone to demosaicking artifacts and noise (Figure 12b). By adapting the merge kernel of the reference frame to the accumulated robustness (Figure 12c), fine details such as the bicycle frame remain visible and merge artifacts disappear. Note that for illustration purposes the images are not post-processed using [8], which would completely remove the color flutter of the artifacts.

3.5 Impact of the Adaptive Fusion Kernel

In this paper, we have introduced the method under the scope of the local-means approach of Equation (3). We have then detailed the steerable kernels of [25] in the subsequent section, explaining how to leverage the 2D local structure tensors to shape the support of the merging kernels. Yet, in Equation (3) we can use any kernel. We compare in Figure 13 the steerable kernels of [25] with the ones proposed by Thibaud Briand in his thesis [4, Chapter 9]. In our formalism, it corresponds to take the 2×2 identity matrix scaled by $1/2$ for Ω. It is thus a structure-agnostic kernel. Figure 13 shows that the approach of Briand [4] leads to blurry results, whereas the steerable kernels gracefully adapt to the local geometry, preventing the blurrying of the edges, thus yielding a much sharper image. This validates the endeavor of shaping more complex merging kernels for efficient image super-resolution.



(a) Reference. (b) Briand [4]. (c) Ours.

Figure 13: Comparison of locally adaptive anisotropic kernel and the general isotropic kernel from [4] for $\times 2$ zoom from a burst of $N = 20$ raw images. Our kernel is narrower than the isotropic counterpart for punctual details like the chimneys, allowing for sharper details. For texture-free areas like the sky, our kernel gets wider than that of [4], allowing for a more efficient denoising. By stretching on edges, our kernel is less prone to create blue and red stripes artifacts on edges. The images are not sharpened for better highlighting the disparity of the remaining blur straight after image fusion.

4 Conclusion

In this paper, we presented a detailed implementation of the raw burst super-resolution algorithm used in the Google Pixel 3 camera. It consists of two successive stages: registration of mosaicked images followed by fusion on a fine RGB grid.

The registration adopts a translation model on non-overlapping patches and is done in two steps. A first pyramid block-matching algorithm predicts a coarse patchwise flow with pixel accuracy. This flow is further refined with three iterations of the inverse compositional algorithm. A qualitative analysis suggests that running these two algorithms on grayscale low-pass filtered variants of the input raw images enhances the accuracy of the flow estimates.

The fusion takes the form a normalized convolution where each frame is sequentially included in both the numerator and the denominator. Each frame's contribution is weighted with respect to structural and robustness coefficients that may reject ill-aligned frames to prevent fusion artifacts. The algorithm returns a single RGB high-resolution version of the reference low-resolution frame in the input raw photograph burst.

Throughout this presentation, we have strove to interpret and propose a coherent algorithm with all possible details to reproduce the results of the original paper. In particular, we filled the blanks for many missing implementation details. Concerning registration, we have robustly aligned raw images via FFT low-pass filtering [4, Chapter 8], have successfully modeled the optical flow with local translations on non-overlapped patches, and have estimated its parameters with ICA iterations [2]. For merging the images, we have handled the brightness-dependent noise level with a generalized Anscombe transform [17] for computing accurate merging kernels, we have corrected the computation of the directional standard deviation coefficients shaping the local merging kernels that were incorrect in the original paper, and have adapted on-the-fly the merging kernels to the case where only the reference frame actually contributes to the merging scheme. We have finally connected the work of Wronski et al. [25] with that of Thibaud Briand in his thesis [4, Chapter 9], showing that his general isotropic kernel and the structure-aware merging kernels presented in our article are two particular instances of the same super-resolution algorithm via normalized convolution. We have empirically shown that the data-adaptive approach of Wronski et al. is better, as expected, and that it competes with recent state-of-the-art CNNs while being as fast and much more computationally efficient, despite our non-optimized implementation.

We believe that this publicly available implementation of the acclaimed paper by Wronski et al. , the first to do so to our knowledge, and the many implementation details we have disclosed in this IPOL article, will benefit the image processing and computational photography communities for deriving new lightweight and fast image and video super-resolution algorithms.

Aknowledgements

This work was partly financed by the DGA Astrid Maturation project SURECAVI ANR-21-ASM3-0002, the Office of Naval research grant N00014-17-1-2552, and the ANR project IMPROVED ANR-22-CE39-0006-04. This work was performed using HPC resources from GENCI–IDRIS (grant 2022-AD011012453R1). Centre Borelli is also a member of Université Paris Cité, SSA and INSERM. We thank Thomas Romera, Thibaud Ehret and Jean-Michel Morel for the insightful discussions throughout the project.

Image Credits

The images in Figures 4, 8, 9 come from the test set of [3]. The image in Figures 5, 11, 12, 13 comes from the "Friant" quantitative example of [15]. The image in Figure 10 comes from the "Rue4" quantitative example of [15].

References

[1] S. Baker and T. Kanade, *Limits on Super-Resolution and How to Break Them*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (2002), pp. 1167–1183. https://doi.org/10.1109/TPAMI.2002.1033210.

[2] S. Baker and I. A. Matthews, *Lucas-Kanade 20 Years On: A Unifying Framework*, International Journal on Computer Vision, 56 (2004), pp. 221–255. https://doi.org/10.1023/B:VISI.0000011205.11775.fd.

[3] G. Bhat, M. Danelljan, L. V. Gool, and R. Timofte, *Deep Burst Super-Resolution*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 9209–9218. https://doi.org/10.1109/CVPR46437.2021.00909.

[4] T. Briand, *Image Formation from a Large Sequence of RAW Images: Performance and Accuracy. (Formation D'image À Partir D'une Grande Séquence D'images RAW: Performance et Précision)*, PhD thesis, University of Paris-Est, France, 2018. https://www.theses.fr/2018PESC1017.

[5] N. A. Dodgson, *Quadratic Interpolation for Image Resampling*, IEEE Transactions on Image Processing, 6 (1997), pp. 1322–1326. https://doi.org/10.1109/83.623195.

[6] DxOMark, *Smartphones Vs Cameras: Closing the Gap on Image Quality*, 2020. https://www.dxomark.com/smartphones-vs-cameras-closing-the-gap-on-image-quality/.

[7] T. Eboli, J. Morel, and G. Facciolo, *Breaking Down Polyblur: Fast Blind Correction of Small Anisotropic Blurs*, Image Processing On Line, 12 (2022), pp. 435–456. https://doi.org/10.5201/ipol.2022.405.

[8] ——, *Fast Two-Step Blind Optical Aberration Correction*, in European Conference on Computer Vision (ECCV), Springer, 2022, pp. 693–708. https://doi.org/10.1007/978-3-031-20068-7_40.

[9] A. Foi, M. Trimeche, V. Katkovnik, and K. O. Egiazarian, *Practical Poissonian-Gaussian Noise Modeling and Fitting for Single-Image Raw-Data*, IEEE Transactions on Image Processing, 17 (2008), pp. 1737–1754. https://doi.org/10.1109/TIP.2008.2001399.

[10] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, *Deep Joint Demosaicking and Denoising*, ACM Transactions on Graphics, 35 (2016), pp. 191:1–191:12. https://dl.acm.org/doi/10.1145/2980179.2982399.

[11] M. Haris, G. Shakhnarovich, and N. Ukita, *Recurrent Back-Projection Network for Video Super-Resolution*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 3897–3906. https://doi.org/10.1109/CVPR.2019.00402.

[12] C. G. Harris and M. Stephens, *A Combined Corner and Edge Detector*, in Alvey Vision Conference, 1988, pp. 23.1–23.6. http://dx.doi.org/10.5244/C.2.23.

[13] S. W. Hasinoff, D. Sharlet, R. Geiss, A. Adams, J. T. Barron, F. Kainz, J. Chen, and M. Levoy, *Burst Photography for High Dynamic Range and Low-Light Imaging on Mobile Cameras*, ACM Transactions on Graphics, 35 (2016), pp. 192:1–192:12. https://dl.acm.org/doi/10.1145/2980179.2980254.

[14] H. Knutsson and C. Westin, *Normalized and Differential Convolution*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1993, pp. 515–523. https://doi.org/10.1109/CVPR.1993.341081.

[15] B. Lecouat, T. Eboli, J. Ponce, and J. Mairal, *High Dynamic Range and Super-Resolution from Raw Image Bursts*, ACM Transactions on Graphics, 41 (2022), pp. 38:1–38:21. https://doi.org/10.1145/3528223.3530180.

[16] O. Liba, K. Murthy, Y. Tsai, T. Brooks, T. Xue, N. Karnad, Q. He, J. T. Barron, D. Sharlet, R. Geiss, S. W. Hasinoff, Y. Pritch, and M. Levoy, *Handheld Mobile Photography in Very Low Light*, ACM Transactions on Graphics, 38 (2019), pp. 164:1–164:16. https://doi.org/10.1145/3355089.3356508.

[17] M. J. Mäkitalo and A. Foi, *Optimal Inversion of the Generalized Anscombe Transformation for Poisson-Gaussian Noise*, IEEE Transactions on Image Processing, 22 (2013), pp. 91–103. https://doi.org/10.1109/TIP.2012.2202675.

[18] S. Mallat, *A Wavelet Tour of Signal Processing - The Sparse Way*, Academic Press, 2009. ISBN 978-0123743701.

[19] H. S. Malvar, L. He, and R. Cutler, *High-Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images*, in IEEE International Conference on Acoustics, Speech and Signal Processing, 2004, pp. 485–488. https://doi.org/10.1109/ICASSP.2004.1326587.

[20] D. Menon, S. Andriani, and G. Calvagno, *Demosaicing With Directional Filtering and a Posteriori Decision*, IEEE Transactions on Image Processing, 16 (2007), pp. 132–141. https://doi.org/10.1109/TIP.2006.884928.

[21] A. Monod, J. Delon, and T. Veit, *An Analysis and Implementation of the HDR+ Burst Denoising Method*, Image Processing On Line, 11 (2021), pp. 142–169. https://doi.org/10.5201/ipol.2021.336.

[22] J. Sánchez, *The Inverse Compositional Algorithm for Parametric Registration*, Image Processing On Line, 6 (2016), pp. 212–232. https://doi.org/10.5201/ipol.2016.153.

[23] H. Takeda, S. Farsiu, and P. Milanfar, *Kernel Regression for Image Processing and Reconstruction*, IEEE Transactions on Image Processing, 16 (2007), pp. 349–366. https://doi.org/10.1109/TIP.2006.888330.

[24] R. Y. Tsai and T. S. Huang, *Multiframe Image Restoration and Registration*, Advance Computer Visual and Image Processing, 1 (1984), pp. 317–339.

[25] B. Wronski, I. Garcia-Dorado, M. Ernst, D. Kelly, M. Krainin, C. Liang, M. Levoy, and P. Milanfar, *Handheld Multi-Frame Super-Resolution*, ACM Transactions on Graphics, 38 (2019), pp. 28:1–28:18. https://dl.acm.org/doi/10.1145/3306346.3323024.