Published in Image Processing On Line on 2018–07–23.
Submitted on 2018–03–27, accepted on 2018–07–16.
ISSN 2105–1232 © 2018 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.2018.224

Video Denoising with Optical Flow Estimation

Antoni Buades, Jose-Luis Lisani

Universitat Illes Balears, Spain
toni.buades@uib.es, joseluis.lisani@uib.es

Communicated by Marc Lebrun *Demo edited by* Jose-Luis Lisani

Abstract

In this paper we describe the implementation of state-of-the-art video denoising algorithm SPTWO [A. Buades, J.L. Lisani, M. Miladinović, Patch Based Video Denoising with Optical Flow Estimation, IEEE Transactions on Image Processing 25 (6), 2573–2586]. This algorithm, inspired by image fusion techniques, uses motion compensation by regularized optical flow methods, which permits robust patch comparison in spatiotemporal volumes. Groups of similar patches are denoised using Principal Component Analysis, which ensures the correct preservation of fine texture and details.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1].

Keywords: video denoising; white Gaussian noise; motion compensation; optical flow

1 Introduction

Most video denoising methods assume the white Gaussian noise model. Let us denote the image sequence by $I(x, y, t)$, with (x, y) the spatial coordinates and t the temporal component, then this model assumes that

$$I(x, y, t) = I_0(x, y, t) + n(x, y, t),$$

where I_0 is the true image sequence and $n(x, y, t)$ the noise i.i.d. realizations of a Gaussian variable of zero mean and standard deviation σ.

Early methods for video denoising simply extended the techniques developed for single images to a sequence of frames. Besides the obvious use of these techniques on a frame by frame basis, local average methods, such as the bilateral filter [19], or patch based methods such as NL-means [4] or BM3D [7] and NLBayes [12] can be easily adapted to video just by extending the neighboring area to the adjacent frames. Kervrann and Boulanger [3] extended NL-means to video by growing

[1]https://doi.org/10.5201/ipol.2018.224

adaptively the spatio-temporal neighborhood. Arias et al. extended NL-Bayes [12] to video [1, 2]. The BM3D extension, VBM4D [14], exploits the mutual similarity between 3-D spatio-temporal volumes constructed by tracking blocks along trajectories defined by the motion vectors. Methods based on sparse decompositions are extended to image sequences [15, 17, 20, 13], as well as approaches based on low rank approximation [9, 10].

The performance of local average methods is improved by introducing motion compensation. These compensated filters estimate explicitly the motion of the sequence and compensate the neighborhoods yielding stationary data [16]. In [5] the authors proposed to combine optical flow estimation and patch based methods for denoising. Their algorithm tends to a fusion of the neighboring frames in the absence of occlusions and a dense temporal sampling. In this ideal scenario, an optical flow or global registration is able to align the frames and fusion is achieved by simple averaging [8]. The algorithm in [5] compensates the failure of these requirements by introducing spatiotemporal patch comparison and denoising in an adapted PCA based transform.

In this paper we describe (Section 2) the method in [5] (the so-called SPTWO algorithm), we discuss its parameters (Section 3) and computational complexity (Section 4) and we illustrate its performance showing some experimental results (Section 5). Finally, some conclusions are exposed in Section 6.

2 Complete Algorithm Description

The proposed algorithm is applied in two steps (see Algorithm 1). The denoised sequence from the first step is used in the second step to improve the final denoising result. Algorithm 2 describes the denoising method, which is applied in a frame by frame basis.

Algorithm 1: SPTWO Denoising Algorithm

```
  Input   : noisy video sequence $\mathcal{I} = I_1, \dots, I_N$ ($I_i(\boldsymbol{x}) \in \mathbb{R}^d$, $d = 1$ for gray images, $d = 3$
            for color images), noise standard deviation $\sigma$
  Output  : denoised sequence $\tilde{\mathcal{I}} = \tilde{I}_1, \dots, \tilde{I}_N$.
1 $\mathcal{O} = \text{SPTWO\_single\_step}(\mathcal{I}, \sigma)$ //First iteration (Algorithm 2)
2 $\tilde{\mathcal{I}} = \text{SPTWO\_single\_step}(\mathcal{I}, \mathcal{O}, \sigma)$ //Second iteration, with oracle (Algorithm 2)
```

Frames alignment. Given a frame I_k from a sequence $\{I_1, I_2, \cdots, I_N\}$, a set of M neighboring frames is first extracted and aligned with respect to I_k (Algorithm 3). This alignment implies the computation of the optical flow[2] between each of the neighboring frames and I_k, and the application of the computed flow to warp each frame into the same spatial domain than I_k. The warping procedure is implemented using bicubic interpolation. Ideally, at the end of the process, all the values in the warped sequence associated to a given pixel $\boldsymbol{x}$, should correspond to the same spatial location in the scene. If registration was accurate and the sequence free of occlusions, a temporal average in this aligned data would be optimal. However, the noise standard deviation would slowly decrease as $1/\sqrt{M}$, where M is the number of aligned frames. Generally, this will not be the case, inaccuracies and errors in the computed flow and the presence of occlusions make this temporal average likely to blur the sequence and have artifacts near occlusions. The proposed approach tends to solve these limitations.

[2]We use the method and code described in [18] to compute the optical flow.

Algorithm 2: SPTWO_single_step

```
Input : noisy video sequence $\mathcal{I} = I_1, \dots, I_N$, oracle video sequence $\mathcal{O} = O_1, \dots, O_N$
        (optional) , noise standard deviation $\sigma$
Output : denoised sequence $\tilde{\mathcal{I}} = \tilde{I}_1, \dots, \tilde{I}_N$.
1  for $k = 1, \cdots, N$ do
       //Get aligned sequence and detect occluded pixels in temporal neighborhood
2      if $\mathcal{O}$ then
3          $[\mathcal{I}^W, \mathcal{O}^W, \mathcal{M}]$ = Align_Frames($\mathcal{I}, \mathcal{O}, k, \sigma$) //Algorithm 3
4      else
5          $[\mathcal{I}^W, \mathcal{M}]$ = Align_Frames($\mathcal{I}, k, \sigma$) //Algorithm 3
       //Denoise frame $k$
6      $\tilde{I}_k = 0$ //Initialize denoised frame
7      $Counter = 0$ //Initialize counter:  for each pixel, count in how many
       denoised patches it is contained
8      Label all the pixels in the domain of $I_k$ as "not processed"
9      for each pixel $\boldsymbol{x}$ in the domain of $I_k$ labeled as "not processed" do
10         if $\mathcal{O}$ then
11             $(\mathcal{S}, \mathcal{S}^{\mathcal{O}})$ = Get_Similar_Patches($\mathcal{I}^W, \mathcal{O}^W, \mathcal{M}, \boldsymbol{x}, k$) //Algorithm 5
12             $\tilde{\mathcal{S}}$ = Patches_Denoising($\mathcal{S}, \mathcal{S}^{\mathcal{O}}, \sigma$) //Denoised patches, Algorithm 7
13         else
14             $\mathcal{S}$ = Get_Similar_Patches($\mathcal{I}^W, \mathcal{M}, \boldsymbol{x}, k$) //Algorithm 5
15             $\tilde{\mathcal{S}}$ = Patches_Denoising($\mathcal{S}, \sigma$) //Denoised patches, Algorithm 7
           //Aggregation
16         for each denoised patch $\tilde{P}$ in $\tilde{\mathcal{S}}$ extracted from frame $k$ do
17             Label as "processed" the center of the patch
18             for each pixel $\boldsymbol{y}$ in $\tilde{P}$ do
19                 $\tilde{I}_k(\boldsymbol{y}) = \tilde{I}_k(\boldsymbol{y}) + \tilde{P}(\boldsymbol{y})$ //Accumulate denoised value at $\boldsymbol{y}$
20                 $Counter(\boldsymbol{y}) = Counter(\boldsymbol{y}) + 1$ //Increase counter
21     for each pixel $\boldsymbol{x}$ in the domain of $I_k$ do
22         $\tilde{I}_k(\boldsymbol{x}) = \tilde{I}_k(\boldsymbol{x})/Counter(\boldsymbol{x})$ //Normalize accumulated values
```

Occlusions management. In order to take into account the presence of occlusions and flow inaccuracies, an occlusion mask $\mathcal{M}$ is computed that indicates in which pixels of the spatial domain the ideal situation holds (Algorithm 4). Occlusions are detected depending on the divergence of the computed flow: negative divergence values indicate occlusions. Additionally, the color difference is checked after flow compensation. A large difference indicates occlusion, or at least failure of the color constancy assumption. We combine both criteria for a pixel $\boldsymbol{x} = (x, y)$ and the computed flow between I_0 and I_1, into the weighting function

$$w(\boldsymbol{x}) = e^{\frac{-|I_0(\boldsymbol{x}) - I_1(\boldsymbol{x}+\boldsymbol{u}(\boldsymbol{x}))|^2}{\sigma_i^2}} \cdot e^{\frac{-min(\mathrm{div}u, 0)^2}{\sigma_d^2}}, \tag{1}$$

where σ_d is fixed while σ_i depends on the noise standard deviation ($\sigma_i = f_o\sigma$, where f_o is a tuning parameter). This weight function is binarized by thresholding at 0.5, giving a mask of occluded pixels (a '0' value meaning "occluded", and '1' "non-occluded"). These occluded pixels having a

Algorithm 3: Align_Frames

```
Input      : video sequence $\mathcal{I} = I_1, \dots, I_N$, oracle video sequence (optional)
             $\mathcal{O} = O_1, \dots, O_N$, index of reference frame $k$, noise standard deviation $\sigma$
Parameters: radius of temporal neighborhood $t$ ($M = 2t+1$ frames in neighborhood)
Output     : set of aligned frames $\mathcal{I}^W$, set of aligned oracle frames (if $\mathcal{O}$ provided) $\mathcal{O}^W$
             occlusion masks $\mathcal{M}$.
1  $\mathcal{I}^W = \emptyset$, $\mathcal{M} = \emptyset$
2  if $\mathcal{O}$ then $\mathcal{O}^W = \emptyset$  $N_k^t = \{j \in 1, \cdots, N / |j-k| \leq t\}$ //Temporal neighborhood of frame $k$
3  for $j \in N_k^t$ do
4      if $j \neq k$ then
5          if $\mathcal{O}$ then
6              $\boldsymbol{u}_{kj}$ = Optical_Flow($O_k, O_j$) //Optical flow from $O_k$ to $O_j$:  $O_k(\boldsymbol{x}) \simeq O_j(\boldsymbol{x} + \boldsymbol{u}_{kj}(\boldsymbol{x}))$
7          else
8              $\boldsymbol{u}_{kj}$ = Optical_Flow($I_k, I_j$) //Optical flow from $I_k$ to $I_j$:  $I_k(\boldsymbol{x}) \simeq I_j(\boldsymbol{x} + \boldsymbol{u}_{kj}(\boldsymbol{x}))$
9      else
10         $\boldsymbol{u}_{kk} = 0$
       //Warp (align) frame $I_j$ using the computed flow
11     for each pixel $\boldsymbol{x}$ in the domain of $I_j$ do
12         $I_j^W(\boldsymbol{x}) = I_j(\boldsymbol{x} + \boldsymbol{u}_{kj}(\boldsymbol{x}))$
13     if $\mathcal{O}$ then
           //Warp (align) frame $O_j$ using the computed flow
14         for each pixel $\boldsymbol{x}$ in the domain of $I_j$ do
15             $O_j^W(\boldsymbol{x}) = O_j(\boldsymbol{x} + \boldsymbol{u}_{kj}(\boldsymbol{x}))$
       //Build occlusion mask (Algorithm 4)
16     if $\mathcal{O}$ then
17         $M_j$ = Occlusion_Mask($I_k, I_j, O_k, O_j, \boldsymbol{u}_{kj}, \sigma$)
18     else
19         $M_j$ = Occlusion_Mask($I_k, I_j, \boldsymbol{u}_{kj}, \sigma$)
20     $\mathcal{M} = \mathcal{M} \cup M_j$
21     $\mathcal{I}^W = \mathcal{I}^W \cup I_j^W$
22     if $\mathcal{O}$ then $\mathcal{O}^W = \mathcal{O}^W \cup O_j^W$
```

negative divergence of the flow and a large color difference after flow compensation are located near the discontinuities of the motion field. The left-right coherence of the computed flow is also checked: ideally, for a given pixel, the flow from frame k to frame j should have the same magnitude and opposite direction than the flow from frame j to frame k. Points that fail to satisfy, up to a tolerance factor, this condition are labeled as "occluded" in the occlusion mask.

Selection of similar patches. Once all the frames in the temporal neighborhood have been aligned, a set of patches is associated to each pixel $\boldsymbol{x}$ in the reference image (Algorithms 5 and 6). Let $\{I_{k-t}^W, \cdots, I_{k+t}^W\}$ be the set of adjacent frames to I_k after warping with the computed optical flow, and let M_j be the occlusion mask between frames I_k and I_j^W, $j \in \{k-t, \cdots, k+t\}$. For each $n \times n$ patch P of the reference frame I_k, we consider the patch (3D block) b^{3D} referring to its extension to the temporal dimension, having M times more pixels than the original one (assuming M patches

Algorithm 4: Occlusion_Mask

```
Input      : source image $I_0$, target image $I_1$, oracle source image (optional) $O_0$, oracle
             target image (optional) $O_1$, optical flow from $I_0$ to $I_1$ (or from $O_0$ to $O_1$, if
             provided): $\boldsymbol{u}$, noise standard deviation $\sigma$
Parameters: factor for gray level values $f_o$ ($\sigma_i = f_o\sigma$), factor for divergence values $\sigma_d$, binary
             mask threshold $\tau_{\text{bin}}$, left-right coherence threshold $\tau_{\text{dist}}$
Output     : binary labels for all the pixels of the image domain: "occluded", "not occluded"
//Check for color differences and negative divergence values (Equation (1))
1  for each pixel $\boldsymbol{x}$ in the domain of $I_k$ do
2      $w(\boldsymbol{x}) = e^{\frac{-|I_0(\boldsymbol{x})-I_1(\boldsymbol{x}+\boldsymbol{u}(\boldsymbol{x}))|^2}{\sigma_i^2}} \cdot e^{\frac{-min(\text{div}u,0)^2}{\sigma_d^2}}$
//Check for left-right coherence of the flow
3  if $O_0$ AND $O_1$ then
4      $\boldsymbol{u}' = \text{Optical\_Flow}(O_1, O_0)$ //Optical flow from $O_1$ to $O_0$
5  else
6      $\boldsymbol{u}' = \text{Optical\_Flow}(I_1, I_0)$ //Optical flow from $I_1$ to $I_0$
7  for each pixel $\boldsymbol{x}$ in the domain of $I_k$ do
8      $\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{u}(\boldsymbol{x}) + \boldsymbol{u}'(\boldsymbol{x} + \lfloor\boldsymbol{u}(\boldsymbol{x})\rfloor)$ //$\lfloor\boldsymbol{u}\rfloor$ denotes the floor operator applied on each
       component of $\boldsymbol{u}$
       //Ideally $\boldsymbol{v} = (v_x, v_y) = (0, 0)$.
9      if $(w(\boldsymbol{x}) < \tau_{bin})$ OR $(\max\{|v_x|, |v_y|\} > \tau_{dist})$ then
10         Output label ($\boldsymbol{x}$)= "occluded"
11     else
12         Output label ($\boldsymbol{x}$)= "not occluded"
```

in the temporal neighborhood[3], $M = 2t + 1$), $b^{3D} = \{P_{k-t}, \cdots, P_{k+t}\}$. The algorithm looks for the extended patches (3D blocks) b'^{3D} closest to b^{3D}. These extended patches are centered at frame I_k and the distance is written as

$$d(b^{3D}, b'^{3D}) = \sum_{i\in\{k-t,\cdots,k+t\}} ||P_i - P_i'||^2.$$

If the K' closest 3D blocks were selected, then, as each extended patch contains M 2D image patches, the group would contain $K' \cdot M$ 2D patches of size $n \times n$, from which patches containing occluded pixels should be removed. In order to guarantee a minimum number of patches in the subsequent denoising step, we select the closest 3D blocks until at least K non-occluded 2D patches are obtained. In the second step of the algorithm, since the distances are computed over partially denoised blocks of patches, all the closest 3D blocks whose distance is below a pre-defined threshold are also selected.

Denoising blocks of similar patches. Each set of similar patches is denoised as described in Algorithm 7. First, the average value and standard deviation of the set of patches is computed, for each color channel. If the standard deviation is small, it is decided that the patches come from a

[3] In practice, if the patch centered at the reference pixel $\boldsymbol{x}$ is occluded in some of the frames of the temporal neighborhood, these frames are not used in the construction of the extended patch, resulting in less than M patches in the 3D block.

Algorithm 5: Get_Similar_Patches

```
Input      : set of aligned frames $\mathcal{I}^W$, set of aligned oracle frames (optional) $\mathcal{O}^W$, occlusion
             masks $\mathcal{M}$, pixel location $\boldsymbol{x}$, index of reference frame $k$
Parameters: radius of spatial neighborhood $r$ ($n = 2r + 1$), radius of temporal neighborhood
             $t$, minimum number of output patches $K$, distance factor $f_d$
Output     : set of similar patches $\mathcal{S}$, set of similar oracle patches (if $\mathcal{O}^W$ provided) $\mathcal{S}^\mathcal{O}$
1  $\Omega$=spatial domain of frames in $\mathcal{I}^W$//Same spatial domain for all aligned frames
2  $N_k^t = \{j \in 1, \cdots, N / |j - k| \leq t\}$ //Temporal neighborhood of frame $k$
3  $N_{\boldsymbol{x}}^r = \{\boldsymbol{z} \in \Omega / ||\boldsymbol{x} - \boldsymbol{z}||_\infty \leq r\}$ //Spatial (squared) neighborhood of pixel $\boldsymbol{x}$
   //Get set of valid frames in temporal neighborhood
4  $N_k^V = \emptyset$
5  for $j \in N_k^t$ do
6      if $M_j(\boldsymbol{z_x}) \in \mathcal{M} =$ "not occluded" $\forall \boldsymbol{z_x} \in N_{\boldsymbol{x}}^r$ then
           //Valid frame if patch around $\boldsymbol{x}$ does not contain occluded pixels
7          $N_k^V = N_k^V \cup \{j\}$
8  ($\mathcal{D}^{3D}$, $\mathcal{B}^{3D}$, $\mathcal{O}^{3D}$)=Get_Similar_3D_Blocks($\mathcal{I}^W$, $\mathcal{O}^W$, $\boldsymbol{x}$, $k$, $N_k^V$)//Algorithm 6
   //Sort distances (and associated 3D blocks) in increasing order
9  ($\mathcal{D}^{3D}_{\text{sorted}}$, $\mathcal{B}^{3D}_{\text{sorted}}$, $\mathcal{O}^{3D}_{\text{sorted}}$)=Sort_distances($\mathcal{D}^{3D}$, $\mathcal{B}^{3D}$, $\mathcal{O}^{3D}$)
   //Get output set of patches
10 $\mathcal{S} = \emptyset$, $n_{\text{patches}} = 0$
11 if $\mathcal{O}^W$ then $\mathcal{S}^\mathcal{O} = \emptyset$ //Maximum allowed 3D distance (#pixels patch $= \text{card}(N_{\boldsymbol{x}}^r)$,
   #valid frames $= \text{card}(N_k^V)$)
12 $\tau_{\text{distance3D}}$=$f_d^2$ × number of pixels in patch × number of channels × number of valid frames
   //$d_{\boldsymbol{y}}^{3D}$, $b_{\boldsymbol{y}}^{3D}$ , $o_{\boldsymbol{y}}^{3D}$, $P_{\boldsymbol{y},j}$ and $O_{\boldsymbol{y},j}$ defined in Algorithm 6
13 for $(d_{\boldsymbol{y}}^{3D}, b_{\boldsymbol{y}}^{3D}, o_{\boldsymbol{y}}^{3D}) \in (\mathcal{D}^{3D}_{sorted}, \mathcal{B}^{3D}_{sorted}, \mathcal{O}^{3D}_{sorted})$ do
14     if $n_{patches} < K$ OR $d_{\boldsymbol{y}}^{3D} \leq \tau_{distance3D}$ then
15         for $(P_{\boldsymbol{y},j}, O_{\boldsymbol{y},j}) \in (b_{\boldsymbol{y}}^{3D}, o_{\boldsymbol{y}}^{3D})$ do
               //Use only patches that do not contain occluded pixels
16             if $M_j(\boldsymbol{z_y}) \in \mathcal{M} =$ "not occluded" $\forall \boldsymbol{z_y} \in N_{\boldsymbol{y}}^r$ then
                   //Add 2D patches to output sets
17                 $\mathcal{S} = \mathcal{S} \cup P_{\boldsymbol{y},j}$
18                 if $\mathcal{O}^W$ then $\mathcal{S}^\mathcal{O} = \mathcal{S}^\mathcal{O} \cup O_{\boldsymbol{y},j}$ $n_{\text{patches}} = n_{\text{patches}} + 1$
```

region with uniform color (a "flat" zone). For large values of noise standard deviation some residual noise might be left in flat zones. This is a common issue to image denoising methods using an adaptive basis [12]. The reason is that in flat zones, the selection of patches is taking into account only noise because of the lack of geometry in these parts, which includes a bias in the selection process. In addition, the PCA analysis with pure noise patches and a reduced number of samples might generate principal values larger than expected. We apply the same solution to this issue as proposed in [12]. Whenever a set of patches is detected to be flat up to the noise oscillations, a simple average (per color channel) of all values in the patches is taken instead of computing a PCA model (see Algorithm 8).

If the patches do not belong to a flat zone the denoising algorithm detailed in Algorithm 9 is used. Assuming that K similar 2D patches are used, the PCA analysis of this set looks for the basis of R^{n^2} better explaining its structure in the sense that most of the information describing all patches

```
Algorithm 6: Get_Similar_3D_Blocks
  Input      : set of aligned frames $\mathcal{I}^W$, set of aligned oracle frames (optional) $\mathcal{O}^W$, occlusion
               masks $\mathcal{M}$, pixel location $\boldsymbol{x}$, index of reference frame $k$, set of valid frames in
               temporal neighborhood of reference frame $N_k^V$
  Parameters: radius of spatial neighborhood $r$, radius of spatial search region $s$
  Output     : set of distances between spatio-temporal blocks $\mathcal{D}^{3D}$, set of spatio-temporal
               blocks $\mathcal{B}^{3D}$, set of spatio-temporal oracle blocks (if $\mathcal{O}^W$ provided) $\mathcal{O}^{3D}$
1 $\Omega$=spatial domain of frames in $\mathcal{I}^W$//Same spatial domain for all aligned frames
2 $\mathcal{D}^{3D} = \emptyset$ //Set of distances between spatio-temporal blocks
3 $\mathcal{B}^{3D} = \emptyset$ //Set of spatio-temporal blocks
4 if $\mathcal{O}^W$ then $\mathcal{O}^{3D} = \emptyset$ //Set of spatio-temporal oracle blocks
5 $N_{\boldsymbol{x}}^r = \{\boldsymbol{z} \in \Omega / ||\boldsymbol{x} - \boldsymbol{z}||_\infty \leq r\}$ //Spatial (squared) neighborhood of pixel $\boldsymbol{x}$
6 $N_{\boldsymbol{x}}^s = \{\boldsymbol{z} \in \Omega / ||\boldsymbol{x} - \boldsymbol{z}||_\infty \leq s\}$ //Spatial (squared) search region around pixel $\boldsymbol{x}$
  //For all pixels in search region
7 for $\boldsymbol{y} \in N_{\boldsymbol{x}}^s$ do
8     $d_{\boldsymbol{y}}^{3D} = 0$ //Distance between spatio-temporal blocks centered at $\boldsymbol{x}$ and $\boldsymbol{y}$
9     $b_{\boldsymbol{y}}^{3D} = \emptyset$ //Spatio-temporal blocks centered at $\boldsymbol{y}$
10    if $\mathcal{O}^W$ then $o_{\boldsymbol{y}}^{3D} = \emptyset$ //Spatio-temporal oracle blocks centered at $\boldsymbol{y}$
11    for $j \in N_k^V$ do
12        $N_{\boldsymbol{y}}^r = \{\boldsymbol{z} \in \Omega / ||\boldsymbol{y} - \boldsymbol{z}||_\infty \leq r\}$ //Spatial (squared) neighborhood of pixel $\boldsymbol{y}$
13        $P_{\boldsymbol{x},j} = \{I_j^W(\boldsymbol{z}), \boldsymbol{z} \in N_{\boldsymbol{x}}^r\}$ //Patch centered at $\boldsymbol{x}$ in frame $j$ (1D array)
14        $P_{\boldsymbol{y},j} = \{I_j^W(\boldsymbol{z}), \boldsymbol{z} \in N_{\boldsymbol{y}}^r\}$ //Patch centered at $\boldsymbol{y}$ in frame $j$ (1D array)
15        if $\mathcal{O}^W$ then
16            $O_{\boldsymbol{x},j} = \{O_j^W(\boldsymbol{z}), \boldsymbol{z} \in N_{\boldsymbol{x}}^r\}$ //Patch centered at $\boldsymbol{x}$ in oracle frame $j$
17            $O_{\boldsymbol{y},j} = \{O_j^W(\boldsymbol{z}), \boldsymbol{z} \in N_{\boldsymbol{y}}^r\}$ //Patch centered at $\boldsymbol{y}$ in oracle frame $j$
18        if $\mathcal{O}^W$ then
19            $d_{\boldsymbol{x},\boldsymbol{y},j}^{2D} = ||O_{\boldsymbol{x},j} - O_{\boldsymbol{y},j}||_2^2$ //Squared distance between oracle patches
20        else
21            $d_{\boldsymbol{x},\boldsymbol{y},j}^{2D} = ||P_{\boldsymbol{x},j} - P_{\boldsymbol{y},j}||_2^2$ //Squared distance between patches
22        $d_{\boldsymbol{y}}^{3D} = d_{\boldsymbol{y}}^{3D} + d_{\boldsymbol{x},\boldsymbol{y},j}^{2D}$
23        $b_{\boldsymbol{y}}^{3D} = b_{\boldsymbol{y}}^{3D} \cup P_{\boldsymbol{y},j}$
24        if $\mathcal{O}^W$ then $o_{\boldsymbol{y}}^{3D} = o_{\boldsymbol{y}}^{3D} \cup O_{\boldsymbol{y},j}$
25    $\mathcal{D}^{3D} = \mathcal{D}^{3D} \cup d_{\boldsymbol{y}}^{3D}$
26    $\mathcal{B}^{3D} = \mathcal{B}^{3D} \cup b_{\boldsymbol{y}}^{3D}$
27    if $\mathcal{O}^W$ then $\mathcal{O}^{3D} = \mathcal{O}^{3D} \cup o_{\boldsymbol{y}}^{3D}$
```

is concentrated in a few vectors of the basis. The amount of information that each vector of the basis conveys is coded in the n^2 principal values. That is, by keeping only the coefficients associated to the most important vectors (the ones with highest corresponding principal value) we keep the maximum of information, while discarding coefficients related to less important vectors we remove noise as proposed in [21]. We refer the reader to [11] for a comprehensive review on PCA theory.

The computation of the PCA of a set of patches is equivalent to the Singular Value decomposition (SVD) of the matrix X having K rows and n^2 columns with each selected patch in a different row,

$$X = U\Sigma V^T,$$

where $U\Sigma$ are the coefficients in the new basis, Σ is a diagonal matrix containing the square root of the principal values and each column of V contains a principal vector, that is an element of the new basis. The decision of canceling a coefficient of a certain patch is not taken depending on its magnitude, but the magnitude of the associated principal value. A more robust thresholding is obtained by comparing the principal values to the noise standard deviation and canceling or maintaining the coefficients of all the patches associated to a certain principal direction. The denoised set of patches can be computed as

$$\tilde{X} = FU\Sigma V^T,$$

where F is a $n^2 \times n^2$ diagonal matrix such that $F_{ii} = 1$ if $\Sigma_{ii} > \tau_{\text{PCA}}\sigma$ and zero otherwise. The whole patch is restored in order to obtain the final estimate by aggregation.

Algorithm 7: Patches_Denoising

```
  Input      : set of similar 2D patches $\mathcal{S}$, set of similar 2D oracle patches (optional) $\mathcal{S}^{\mathcal{O}}$,
               noise standard deviation $\sigma$
  Parameters: flat parameter $f_{\text{flat}}$
  Output     : denoised set of patches $\tilde{\mathcal{S}}$
  //Compute average value, per channel, from set $\mathcal{S}$ and average standard
  deviation
1 $(\boldsymbol{\mu}, s) = \text{Average\_Patches}(\mathcal{S})$ //Algorithm 8
2 if $s < f_{flat} \times \sigma$ then
      //All the patches in $\mathcal{S}$ have similar values
3     $\tilde{\mathcal{S}} = \boldsymbol{\mu}$ //Assign the same constant value, per channel
4 else
5     $\tilde{\mathcal{S}} = \text{PCA\_Denoising}(\mathcal{S}, \mathcal{S}^{\mathcal{O}}, \sigma)$ //Algorithm 9
```

Second "oracle" iteration. A second iteration of the algorithm is performed using the "oracle" strategy. Once the whole sequence has been restored, we re-apply the algorithm on the initial noisy sequence, but motion estimation and patch selection are performed on the result of the first iteration.

Let $\{I^W_{k-t}, \cdots, I^W_{k+t}\}$ and $\{I^{0W}_{k-t}, \cdots, I^{0W}_{k+t}\}$ be the warped noisy and initially restored images in a temporal neighborhood of I_k where the optical flow has been computed using initially restored images I^0_k and I^0_j. For each patch P of the reference frame I_k, we consider the extended patches b^{3D} and b^{03D} referring to the extension to the temporal dimension of the patch and its counterpart in the already denoised sequence. The extended patches that will be selected as similar are the ones minimizing the distance

$$d(b^{03D}, b'^{03D}) = \sum_{i \in \{k-t, \cdots, k+t\}} ||P^0_i - P'^0_i||^2.$$

Now we have two different sets containing each one K 2D patches of size $n \times n$. One set is formed by the patches of the noisy sequence and the other one by the corresponding patches of the already denoised sequence.

The PCA is computed in the set of already denoised patches. Let X denote the matrix containing the selected patches of the noisy sequence as rows and X^0 the corresponding matrix with the same patches of the already filtered sequence. We compute the basis associated to X^0 making use of the SVD,

$$X^0 = U^0 \Sigma^0 V^{0T}.$$

Algorithm 8: Average_Patches

```
Input   : set of 2D patches $\mathcal{S}$
Output  : average value $\boldsymbol{\mu}$ (per channel), average standard deviation of the channels $s$
1  $\Omega_{\mathcal{S}}$=spatial domain of patches in $\mathcal{S}$ //All the patches share the same spatial domain
   //Initialize variables
2  if color patches then
3      $V^R = \emptyset$ //Set of red values in $\mathcal{S}$
4      $V^G = \emptyset$ //Set of green values in $\mathcal{S}$
5      $V^B = \emptyset$ //Set of blue values in $\mathcal{S}$
6  else
7      $V = \emptyset$ //Set of gray values in $\mathcal{S}$
8  for $\boldsymbol{x} \in \Omega_{\mathcal{S}}$ do
9      for $P \in \mathcal{S}$ do
10         if color patches then
11             $P^R_{\boldsymbol{x}}$=red value of patch $P$ at pixel $\boldsymbol{x}$
12             $P^G_{\boldsymbol{x}}$=green value of patch $P$ at pixel $\boldsymbol{x}$
13             $P^B_{\boldsymbol{x}}$=blue value of patch $P$ at pixel $\boldsymbol{x}$
14             $V^R = V^R \cup \{P^R_{\boldsymbol{x}}\}$
15             $V^G = V^G \cup \{P^G_{\boldsymbol{x}}\}$
16             $V^B = V^B \cup \{P^B_{\boldsymbol{x}}\}$
17         else
18             $P_{\boldsymbol{x}}$=gray value of patch $P$ at pixel $\boldsymbol{x}$
19             $V = V \cup \{P_{\boldsymbol{x}}\}$
20 if color patches then
21     $\boldsymbol{\mu} = \big(\text{average}(V^R), \text{average}(V^G), \text{average}(V^B)\big)$
22     $s = \text{average}\{\text{standard deviation}(V^R), \text{standard deviation}(V^G), \text{standard deviation}(V^B)\}$
23 else
24     $\boldsymbol{\mu} = \text{average}(V)$
25     $s = \text{standard deviation}(V)$
```

This basis is adapted to the already denoised patches which are noise-free. The coefficients of the noisy patches are computed in this new basis and modified by a Wiener filter before reconstruction. Let A^D be the modified coefficients of the noisy patches, computed as $A^D = F \circ A$, where $A = XV^0$ are the noisy coefficients, $\circ$ denotes the element-wise matrix product and F is a $n^2 \times n^2$ matrix such that $F_{ij} = \frac{(A_{ij})^2}{(A_{ij})^2+\sigma_R^2}$, with $\sigma_R = \tau_{\text{PCA}}\sigma$. Then, the denoised patches are computed as

$$\tilde{X} = A^D {V^0}^T.$$

Color image denoising. Color images are denoised directly without the use of any color decorrelating transform. Each color patch is considered as a vector with three times more components than in the single channel case. The use of several frames makes the number of patches available much larger. This permits the use of PCA with color patches. That is, PCA adapts a color decorrelation transform for each group of patches, thus increasing the effectiveness of the model.

The final color algorithm is applied to vectorial patches, following the steps described in Algorithm 7. The optical flow is in this case computed on the gray version of the color images. Each

Algorithm 9: PCA_Denoising

```
Input      : set of similar 2D patches $\mathcal{S}$, set of similar 2D oracle patches (optional) $\mathcal{S}^{\mathcal{O}}$,
             noise standard deviation $\sigma$
Parameters: denoising threshold $\tau_{\text{PCA}}$
Output     : denoised set of patches $\tilde{\mathcal{S}}$
1  //Notation
2  $K$ = number of patches in $\mathcal{S}$ (= number of patches in $\mathcal{S}^{\mathcal{O}}$)
3  $n^2 = n \times n$ = number of pixels in each patch
4  $N = d \times n^2$ = number of values in each patch //$d = 1$ for gray patches and $d = 3$ for
   color patches
   //Reorganize patches in $\mathcal{S}$ in matrix form
5  $X = K \times N$ matrix containing the patches in $\mathcal{S}$ as row vectors
6  if $\mathcal{S}^{\mathcal{O}}$ then
       //Reorganize patches in $\mathcal{S}^{\mathcal{O}}$ in matrix form
7      $X^O = K \times N$ matrix containing the patches in $\mathcal{S}^{\mathcal{O}}$ as row vectors
   //Get centered patches (their barycenter is the null vector)
8  $\boldsymbol{b}$ = Barycenter of patches in $X$ (row vector)
9  $X_c = X - \mathbf{1} \cdot \boldsymbol{b}$ //$\mathbf{1}$ is a column vector of ones
10 (resp. $X_c^O = X^O - \mathbf{1} \cdot \boldsymbol{b}^{\boldsymbol{O}}$, where $\boldsymbol{b}^{\boldsymbol{O}}$ is the barycenter of the patches in $X^O$)
   //Denoise patches
11 $\sigma_R^2 = (\tau_{\text{PCA}})^2 \times \sigma^2$
12 if $\mathcal{S}^{\mathcal{O}}$ then
13     $X_c^O = U^O \Sigma^O V^{O^T}$ //SVD decomposition of $X_c^O$
14     $A^O = U^O \Sigma^O$ //Coefficients of the patches in $X_c^O$ in the basis $V^O$
15     $A = X_c V^O = U \Sigma$ //Coefficients of the patches in $X_c$ in the basis $V^O$
16     $F = (F_{ij}), \qquad F_{ij} = \frac{(A_{ij})^2}{(A_{ij})^2 + \sigma_R^2}$
17     $A^D = F \circ A$ //Modified coefficients ($\circ$ denotes element-wise matrix product)
18     $X_{cD} = A^D V^{O^T}$ //Denoised patches (centered)
19 else
20     $X_c = U \Sigma V^T$ //SVD decomposition of $X$
21     $A = U \Sigma$ //Coefficients of the patches in $X_c$ in the basis $V$
22     $F$ = diagonal matrix, $\qquad F_{ii} = \begin{cases} 1 & \text{if } \Sigma_{ii}^2 \geq \sigma_R^2 \\ 0 & \text{otherwise} \end{cases}$
23     $A^D = FA$ //Modified coefficients
24     $X_{cD} = A^D V^T$ //Denoised patches (centered)
25 $X_D = \mathbf{1} \cdot \boldsymbol{b} + X_{cD}$ //Denoised patches (undo centering)
26 Store the denoised patches in $\tilde{\mathcal{S}}$
```

channel is resampled with the same flow, and the same mask of occlusions is used for all the channels.

3 Parameters of the Method

Table 1 summarizes the parameters of the proposed denoising method.

- λ, is the main parameter of the optical flow computation. We use the method described in [18]

Table 1: Parameters of the method.

Parameter	Description	Default value
λ	data attachment weight in optical flow computation	1st step: 0.075 2nd step: 0.15
f_o	factor for gray level values in occlusions mask	5.5
σ_d	factor for divergence values in occlusions mask	1
τ_{bin}	threshold for binarization of occlusions mask	0.5
τ_{dist}	threshold for left-right coherence in occlusions mask	1
t	radius of temporal neighborhood	7
r	radius of spatial neighborhood	2
s	radius of spatial search region	12
K	minimum number of patches	Gray images: 55 Color images: 95
f_d	distance factor between 3D blocks	1st step: 0 2nd step: 2
f_{flat}	flat parameter	0.85
τ_{PCA}	threshold for PCA denoising/Wiener filtering	1st step: 1.8 2nd step: 1.45

for this computation, and the rest of its parameters are set to the default values proposed in this article. This parameter determines the smoothness of the obtained flow, the smaller the parameter the smoother the result. In the first step of the denoising algorithm the input data are noisy and the obtained flow will be spatially unstable, for this reason we use a smaller value of the parameter to smooth the result. In the second step, since the flow is computed from denoised data, we use a higher value.

- f_o, this parameter controls the weight of the difference in gray levels in the occlusion mask ($\sigma_i = f_o\sigma$, Equation (1)). As the value of the parameter increases less importance is given to the gray level difference in determining that the flow is unreliable at a given pixel.

- σ_d, this parameter controls the weight of the divergence term in the occlusion mask (Equation (1)). This term is only accounted for when negative values of the divergence occur (since they are associated to the presence of occlusions). In this case, as the value of the parameter increases, less importance is given to the value of the divergence in determining that the flow is reliable at a given pixel.

- τ_{bin}. Equation (1) produces a value in the range $(0, 1]$ associated to the reliability of the flow at a given pixel. Values close to 1 indicate that the flow is highly reliable while values close to 0 indicate the opposite. The parameter τ_{bin} defines the threshold between "reliable" (i.e. occluded) and "unreliable" pixels.

- τ_{dist}, is the threshold for left-right coherence of the computed flow. If the difference between forward and backward flow at a given pixel is less than this threshold it is decided that the flow is reliable at the pixel, and unreliable otherwise.

- t, is the radius of the temporal neighborhood in the vicinity of each frame. This implies that $M = 2 \times t + 1$ frames are aligned and processed every time that a frame is denoised.

- r, is the radius of the (square) patch around each pixel used to perform the denoising process. This implies that each side of the patch has $n = 2 \times r + 1$ pixels, and that the total number of pixels in the patch is $(2 \times r + 1) \times (2 \times r + 1)$.

- s, is the radius of the area around each pixel where similar patches are searched for.
- K, is the minimum number of similar 2D patches needed to perform the denoising process. This number depends on the dimension of the patches. For color images, since the dimension of the patches increases (three times more values are used per pixel), we use a higher value than for gray level images.
- f_d, determines the maximum distance factor between 3D blocks (used only in the second step of the algorithm) below which all the patches in the block are used for denoising.
- f_{flat}, determines the maximum standard deviation of the values of a 3D block of patches below which the patches in the block are considered as "flat" patches.
- τ_{PCA}. In the first step of the method this parameter determines which vectors in the basis obtained from PCA analysis of the set of similar patches are used to reconstruct the denoised patch. The higher the value of this parameter the less vectors are used to reconstruct the final result and the denoising effect is more apparent (at the expense of losing image details and textures). In the second step, the parameter determines the attenuation of each coefficient in the Wiener filtering.

The default values reported in the table were obtained by experimental evaluation and provide the best denoising results over a wide range of test images. These default parameters have been used in the experiments shown in the next section and also in the online demo associated to this paper.

In order to analyze how the variations of the different parameters affect the denoising result, we modify the value of one parameter at each time, while keeping the rest of parameters fixed to their default values. We use for our tests 5 gray sequences composed of 8 frames (sequences *army*, *cars*, *girls*, *statB* and *taxi*, see Figure 4). White Gaussian noise is added to each sequence (we use two different levels of noise in the tests) and the Root Mean Squared Error (RMSE) with respect to the original (noise-free) sequences is computed for the central frame of each denoised sequence. The average of RMSE values over the 5 sequences is displayed in Figures 1 and 2 for each combination of parameters.

We observe (Figure 1 a and b) that the denoising results are quite stable to variations of the parameter λ of the optical flow algorithm, although slightly better results are obtained with the default values. A similar remark applies to the occlusion parameter f_o (Figure 1 c), provided that its value is high enough. Concerning the size of the patches (parameter r, Figure 1 d), if they are too small or too large the denoising results deteriorate. The optimum result is obtained for 5×5 patches ($r = 2$). The results improve when the size of the temporal neighborhood t increases (Figure 1 e). It must be taken into account that a very short sequence (8 frames) was used in the tests and that for $t = 5$ all the frames of the sequence were used for denoising. In practice, $t = 7$ (15 frames) is used as the default parameter since larger values imply higher computation times. In Figure 1 f we observe that an excessive increase in the number of patches K used for denoising may lead to worse results in case of low levels of noise, while it improves the results when the noise level is higher. We use a compromise value of $K = 55$ as default parameter.

In general, the denoising results deteriorate as the flat parameter f_{flat} increases (see Figure 2 a). This is because a high value of the parameter implies that more patches are replaced by a constant value in the denoised frames, producing an excessive smoothing. However, slightly better results are obtained with a relatively high value of the parameter for high levels of noise, which suggests that the denoising method based on PCA is unable to faithfully reconstruct uniform regions of the image when the noise is high. In Figure 2 b we observe that the denoising results are quite stable with respect to parameter f_d. Finally, higher values of the threshold for PCA denoising (parameter τ_{PCA}, Figure 2 c and d) produce better results, when used in the first step of the method. In the second step, the results are quite stable with respect to the parameter.

4 Computational Complexity

It is difficult to give a theoretical estimate of the complexity of the proposed method due to the large number of algorithms involved. Instead, we give an empirical estimate. We proceed as follows: several versions of the same sequence are obtained at different resolutions by zooming out the original frames with decreasing zoom factors; the zoom factors are chosen such that the size (in pixels, i.e. width $\times$ height) of each frame is half the size of the frames in the previous sequence; we add noise to the sequence and apply the proposed denoising method, recording the computation time per frame.

We have chosen for this experiment the sequence *statB* (see Figure 4), composed of 8 gray frames. The initial size of the frames is $740 \times 560 = 414400$ pixels. The obtained result, for two levels of noise ($\sigma = 20$ and $\sigma = 40$) is displayed in Figure 3-left. We observe a linear relation between the computation time and the size of the frames. We also observe that the computation time slightly increases with the level of noise. This is due to the fact that when noise increases less 3D blocks will be considered as “flat” and therefore they will be processed using Algorithm 9, which implies more computations than simply replacing the block by its average value. The reported computation times where obtained with a Intel Xeon CPU X7560 @ 2.27GHz and 32MB RAM.

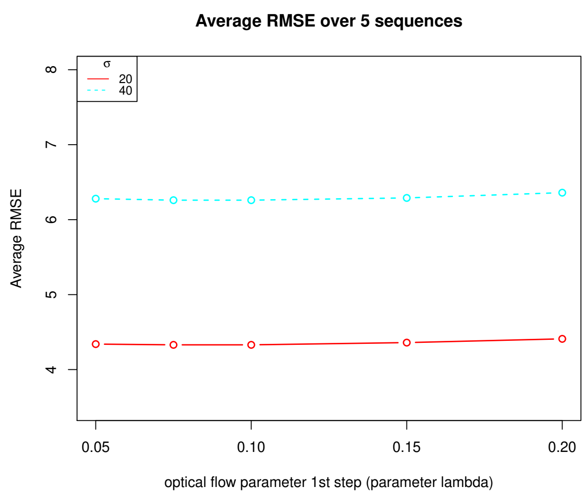
Remark that some parts of the method are easily parallelizable using OMP directives[4]. This is possible in Algorithm 3, where all the neighbor frames of the reference frame can be aligned simultaneously. Parallelism is also used when computing divergences and gradients for the optical flow estimation and for image interpolation when warping the frames. Finally, the denoising of each pixel of a given frame (lines 10 to 20 in Algorithm 2) can also be computed in parallel. The use of OMP directives reduces significantly the computation time, as shown in Figure 3-right. In this case, using the 32 cores of the processor mentioned above, the computation times where reduced by a factor of 20.

5 Experimental Results

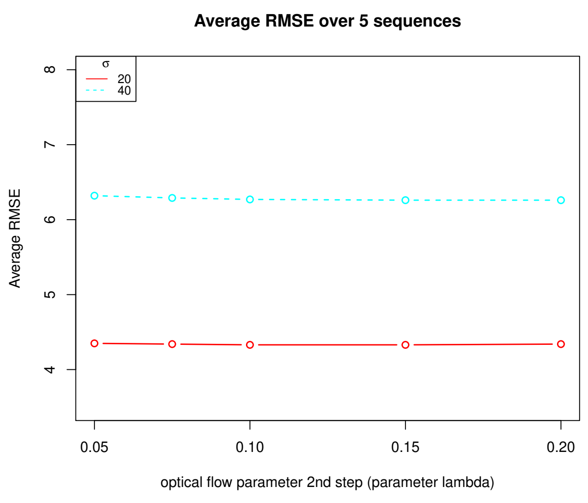
In this section we compare the proposed approach with state-of-the-art algorithms VBM3D [6] and VBM4D [14]. The Matlab implementations of VBM3D and VBM4D were obtained from the author’s web site and the default parameters were used in the tests.

Since VBM4D is the closest algorithm in the literature we briefly discuss the main differences in both the selection and filtering stages. VBM4D uses a motion estimation based on block matching while we use an optical flow estimation. There are two advantages in using optical flow: first the minimization is global and contains a regularization term, which makes the motion selection more independent of noise values. This might not be so important for other applications but it is crucial for denoising. The second difference is that optical flow provides subpixel accuracy, permitting a resampling of the images. This resampling improves the performance of the method. In the filtering stage, VBM4D uses a fixed transform based on bi-orthogonal wavelets and DCT. This is the same transform used by BM3D for images. However, in video we dispose of many more samples and we can take advantage of this fact by learning a better model for denoising. PCA allows us to obtain such an adaptive transform.

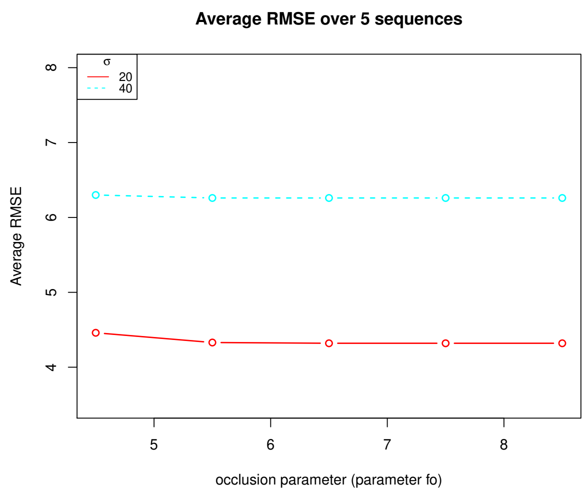
[4]http://www.openmp.org/



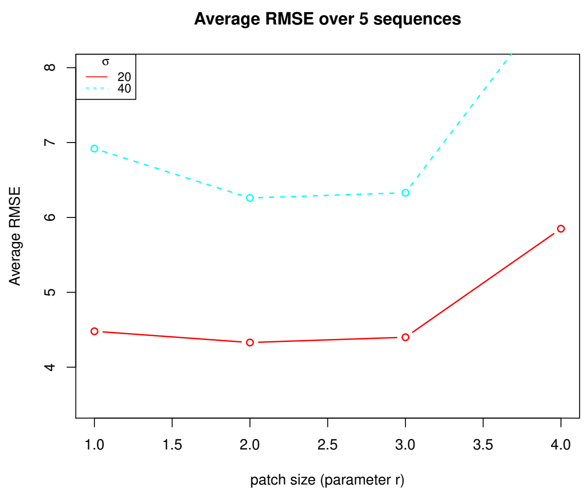
a) λ (1st step)



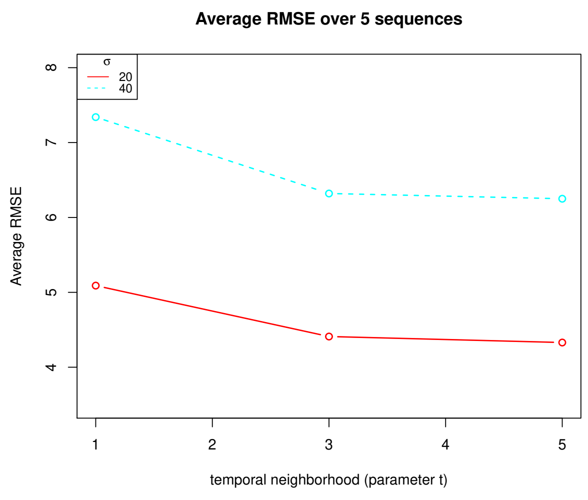
b) λ (2nd step)



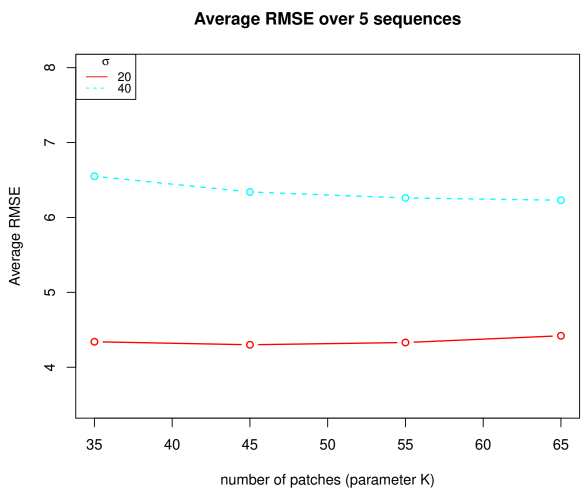
c) f_o



d) r

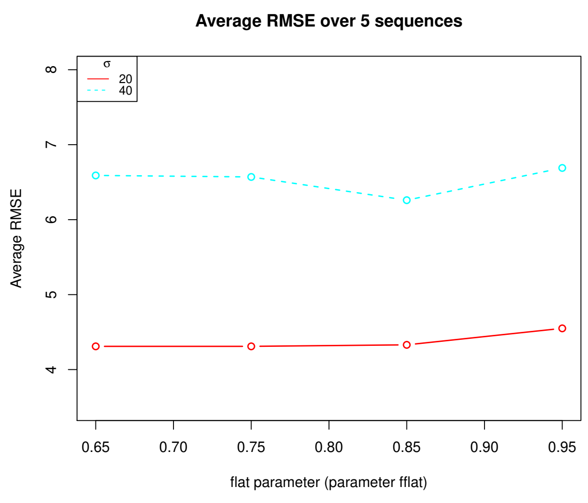


e) t

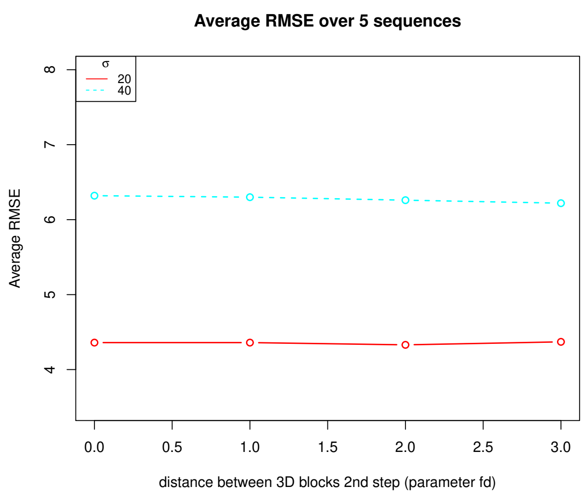


f) K

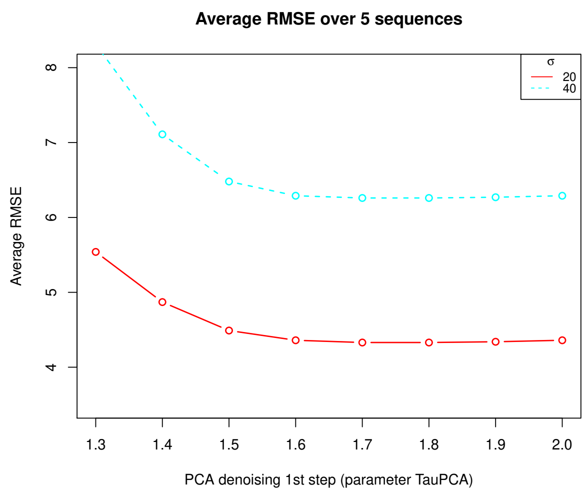
Figure 1: Average RMSE over 5 sequences for varying values of the parameters.



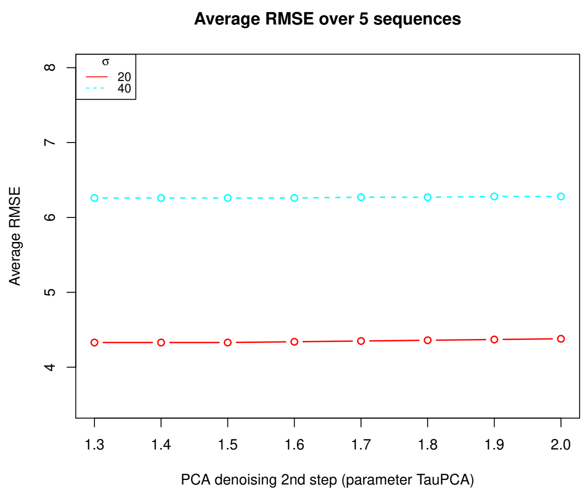
a) f_{flat}



b) f_d



c) τ_{PCA} (1st step)



d) τ_{PCA} (2nd step)

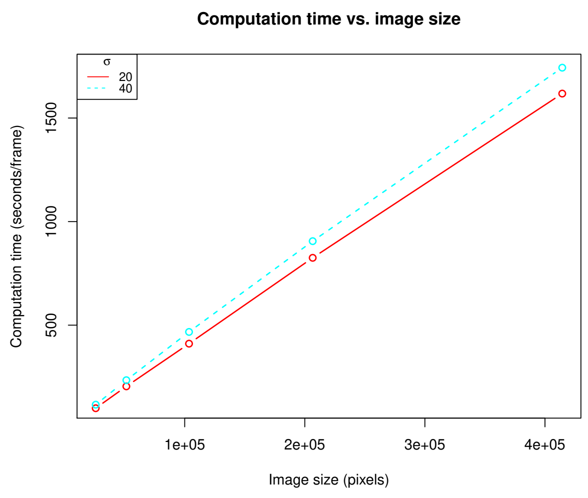
Figure 2: Average RMSE over 5 sequences for varying values of the parameters.

We compare the performance of VBM3D, VBM4D and the proposed algorithm (SPTWO) both visually and numerically in several image sequences. Figure 4 displays the central frame of the used sequences. Some of the sequences are composed of 8 frames while others consist of 30 frames. We apply the proposed method with the default parameters listed in Table 1.

A Gaussian noise with increasing levels of standard deviation ($\sigma \in \{10, 20, 30, 40, 50\}$) was added to the sequences (some examples are displayed in Figure 5), which were denoised using the compared methods.

The Root Mean Squared Error (RMSE) with respect to the original (noise-free) sequences is displayed in Table 2. The values in the table correspond to the RMSE computed for the central frame of each sequence. Moreover, the average of these RMSE values, for each method and each noise level, is also displayed in the last column. Notice that the smaller RMSE values are obtained, in general, with the proposed algorithm. It is interesting to note that the performance of VBM3D is slightly better than VBM4D for low levels of noise (below $\sigma = 40$), while for higher levels of noise VBM4D is slightly better.

Figures 6, 7 and 8 display the denoising results of the compared algorithms. The figures also display the difference of the denoised image with the noisy one, and the difference of the denoised



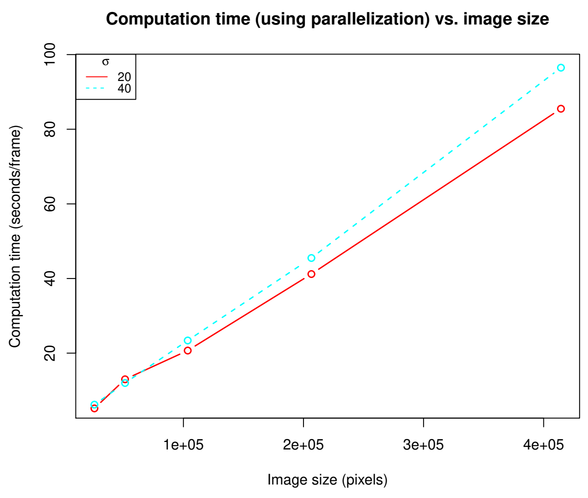


Figure 3: Computation time for increasing image sizes and different noise levels. Left, withouth parallelization of some algorithms. Right, with parallelization.

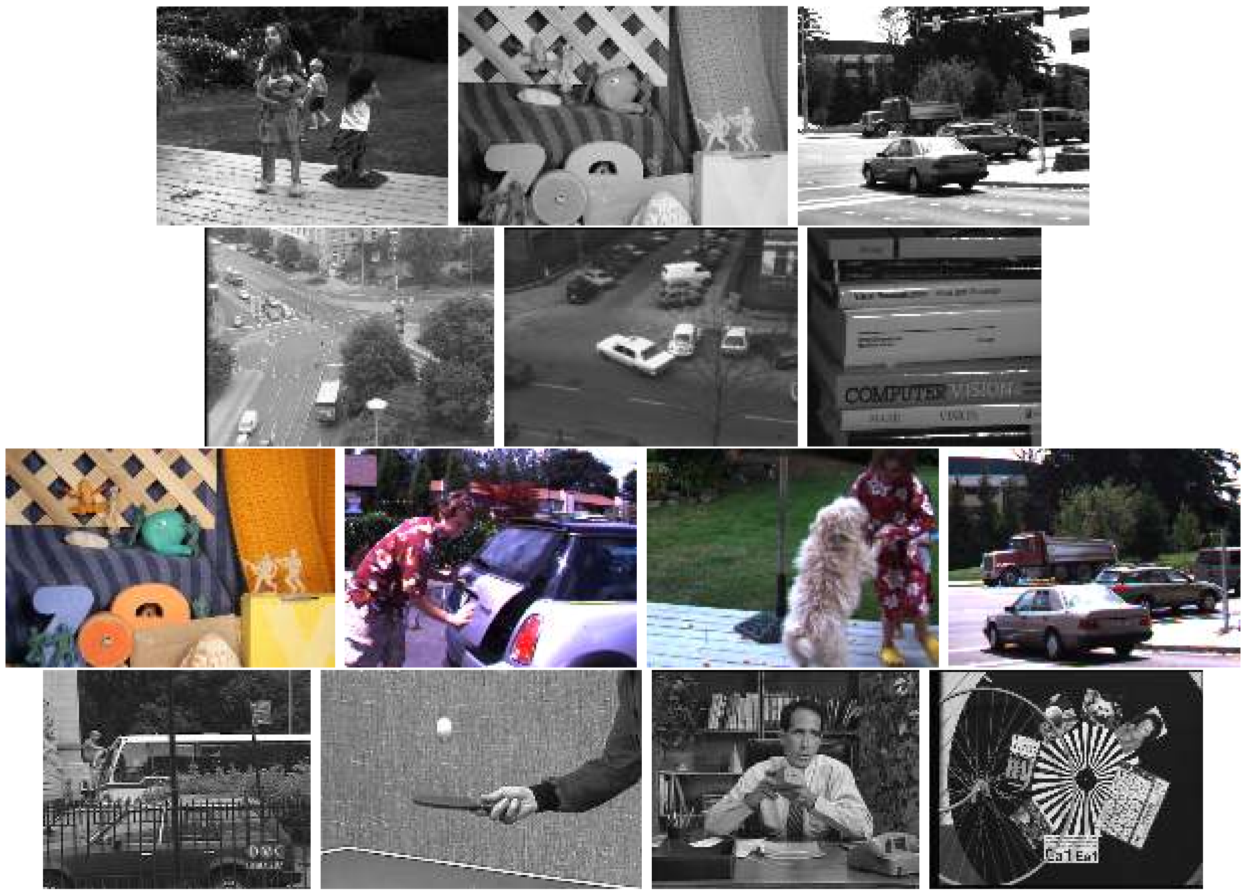


Figure 4: Central frame of the original sequences used in our tests. From left to right and from top to bottom: girls, army, truck, statB, taxi, iseq, army (color), cooper, girls (color), truck (color), gbus, gstennis, gsalesman, gbicycle. The last four sequences are composed of 30 frames, while the rest are composed of 8 frames.

image with the original one. The difference with the noisy image displays the noise removed by each algorithm. The absence of noticeable details in the removed noise should indicate the preservation of all texture and features of the original image. However, as illustrated by the denoised images, this

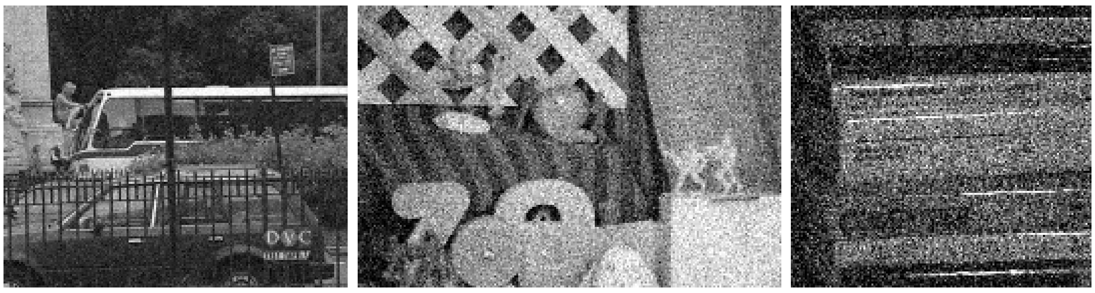


Figure 5: Three examples of noisy sequences (the central frame of the sequence is shown). From left to right: $\sigma = 20$, $\sigma = 30$ and $\sigma = 50$.

	girls	army	truck	statB	taxi	iseq	gbus	gstennis	gsalesman	gbicycle	**average**
$\sigma = 10$											
VBM3D	3.86	3.49	**3.07**	3.11	**3.01**	2.89	5.73	**4.75**	3.92	3.78	3.76
VBM4D	3.66	3.33	3.28	3.16	3.21	2.80	5.62	4.83	3.93	3.88	3.77
SPTWO	**3.53**	**3.08**	3.18	**3.02**	3.03	**2.06**	**4.05**	4.85	**3.85**	**3.52**	**3.41**
$\sigma = 20$											
VBM3D	5.48	4.93	**4.63**	4.22	4.26	4.36	9.03	**7.42**	5.57	5.84	5.57
VBM4D	5.31	4.87	5.05	4.67	4.67	4.26	8.87	7.89	6.01	5.90	5.75
SPTWO	**4.77**	**4.12**	4.65	**3.99**	**4.15**	**2.88**	**6.27**	8.49	**5.46**	**5.66**	**5.04**
$\sigma = 30$											
VBM3D	6.78	6.06	5.96	5.39	5.43	5.71	11.41	**10.88**	7.27	7.54	7.24
VBM4D	6.58	6.02	6.41	5.89	5.93	5.58	11.25	11.84	7.84	7.63	7.50
SPTWO	**5.73**	**4.99**	**5.93**	**4.91**	**5.15**	**3.63**	**8.01**	12.40	**6.68**	**6.96**	**6.44**
$\sigma = 40$											
VBM3D	7.91	6.90	7.18	6.50	6.41	6.70	13.25	**13.45**	8.82	9.10	8.62
VBM4D	7.75	6.85	7.62	6.94	6.89	6.52	13.11	13.96	9.33	9.26	8.82
SPTWO	**6.72**	**5.76**	**7.11**	**5.81**	**5.92**	**4.23**	**9.44**	14.06	**7.77**	**8.37**	**7.51**
$\sigma = 50$											
VBM3D	9.23	7.88	8.69	7.61	7.63	7.63	15.37	14.87	10.73	11.67	10.13
VBM4D	8.75	7.62	8.77	7.81	7.81	7.33	14.81	14.92	10.77	10.65	9.92
SPTWO	**7.61**	**6.50**	**8.27**	**6.75**	**6.78**	**4.92**	**10.75**	**14.56**	**8.81**	**9.63**	**8.45**

Table 2: RMSE results. The values correspond to the RMSE computed for the central frame of each sequence. The average RMSE for each method and each noise level is displayed in the last column.

absence of details in the removed noise does not guarantee that all meaningful information of the original image has been kept. Indeed, the removed structure might be hidden by the noise. For this reason, we also display the image error, actually containing removed information from the original image.

A first visual inspection illustrates that VBM3D and VBM4D perform similarly and its main weakness is the excessive blurring of image details. Not only texture but also geometry may be removed by these approaches. This can be observed in the letters of the book in Figure 8. The proposed algorithm recovers better all image details in all three cases, even in the presence of strong noise.

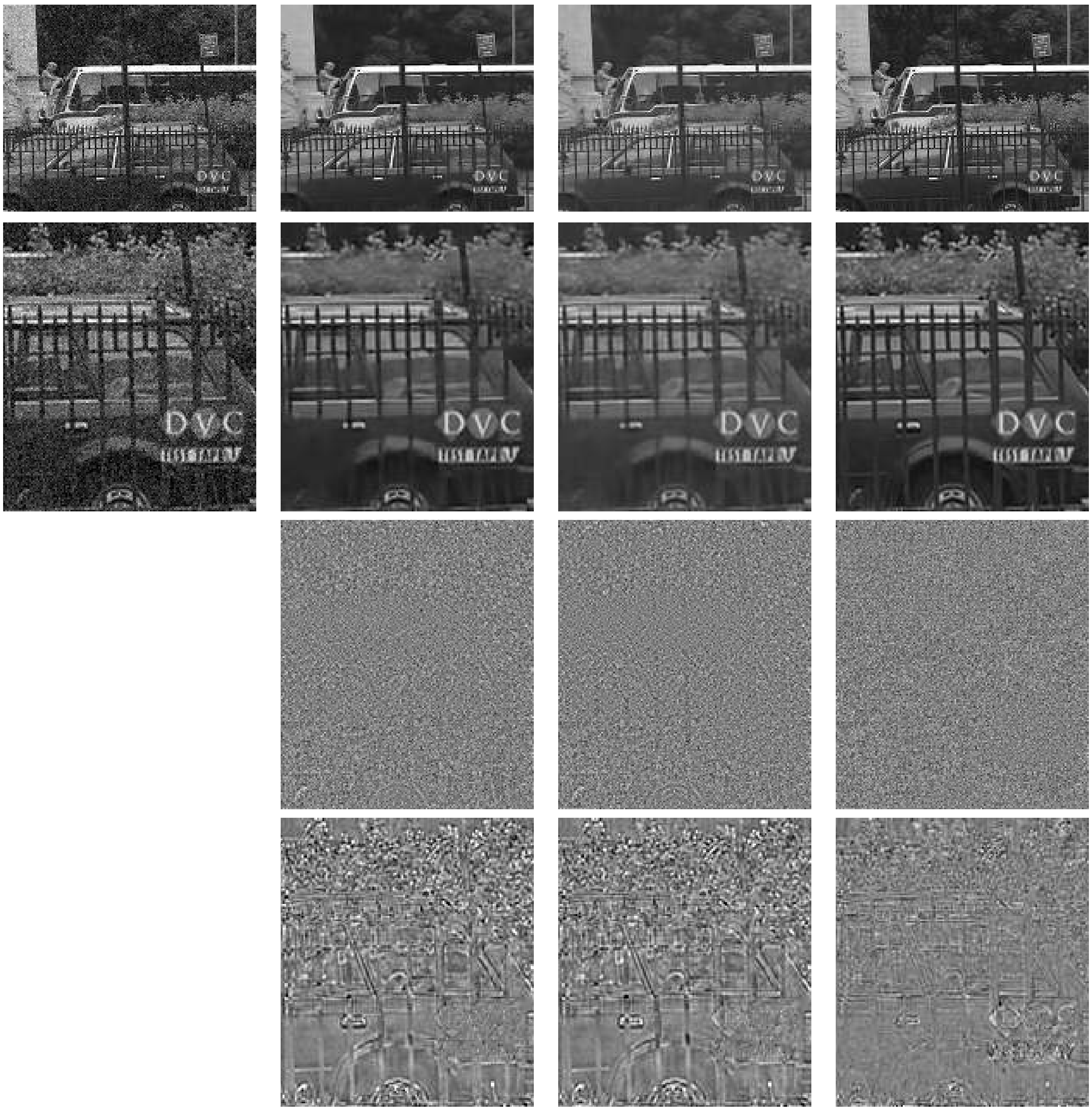


Figure 6: Denoising results for the *gbus* sequence with $\sigma = 20$. First row, from left to right: noisy frame, VBM3D (RMSE=9.03), VBM4D (RMSE=8.87), SPTWO (RMSE=6.27). Second row: detail of the above images. Third row: differences with respect to the noisy frame. Fourth row: differences with respect to the original (noise-free) frame.

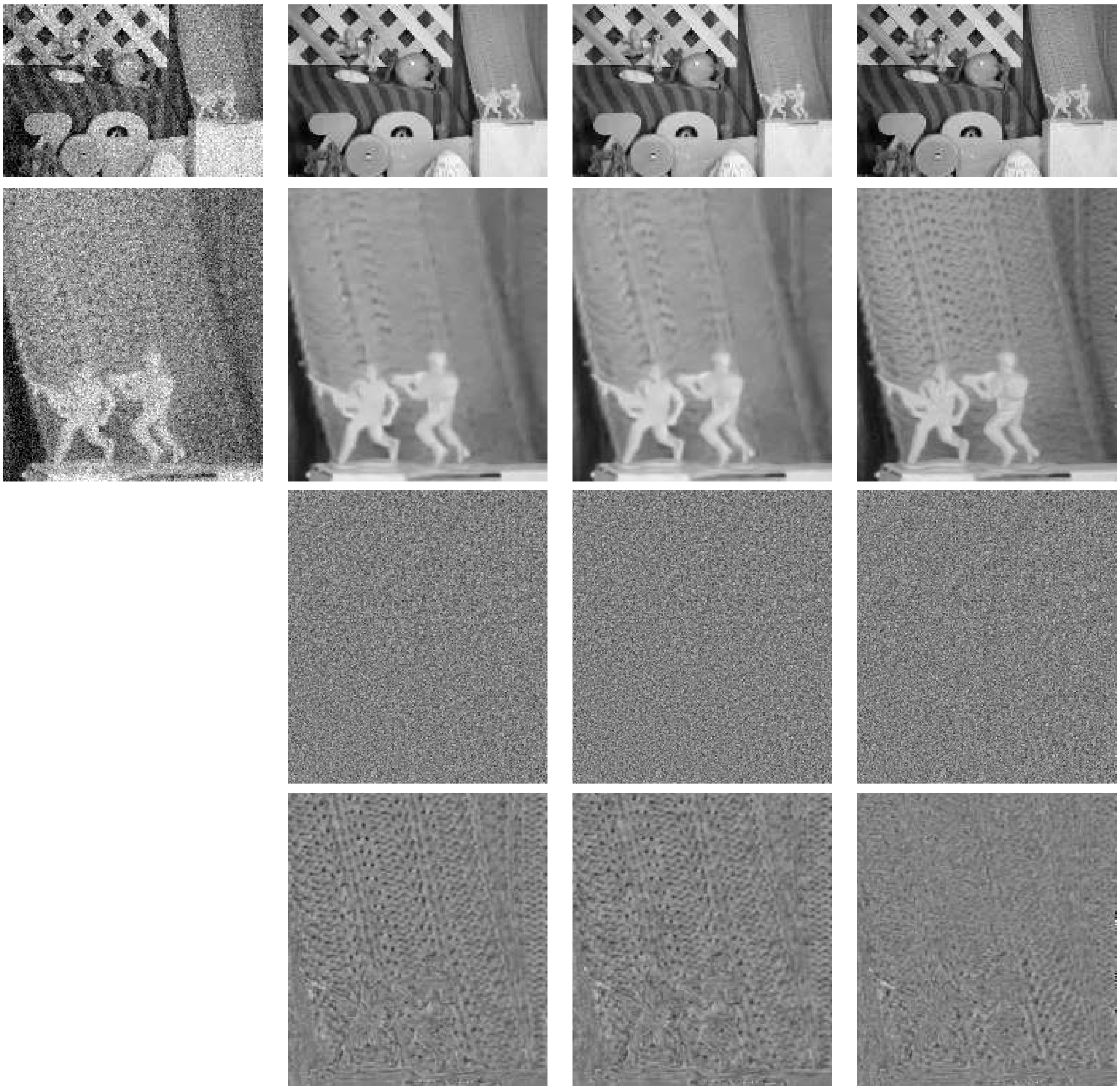


Figure 7: Denoising results for the *army* sequence with $\sigma = 30$. First row, from left to right: noisy frame, VBM3D (RMSE=6.06), VBM4D (RMSE=6.02), SPTWO (RMSE=4.99). Second row: detail of the above images. Third row: differences with respect to the noisy frame. Fourth row: differences with respect to the original (noise-free) frame.

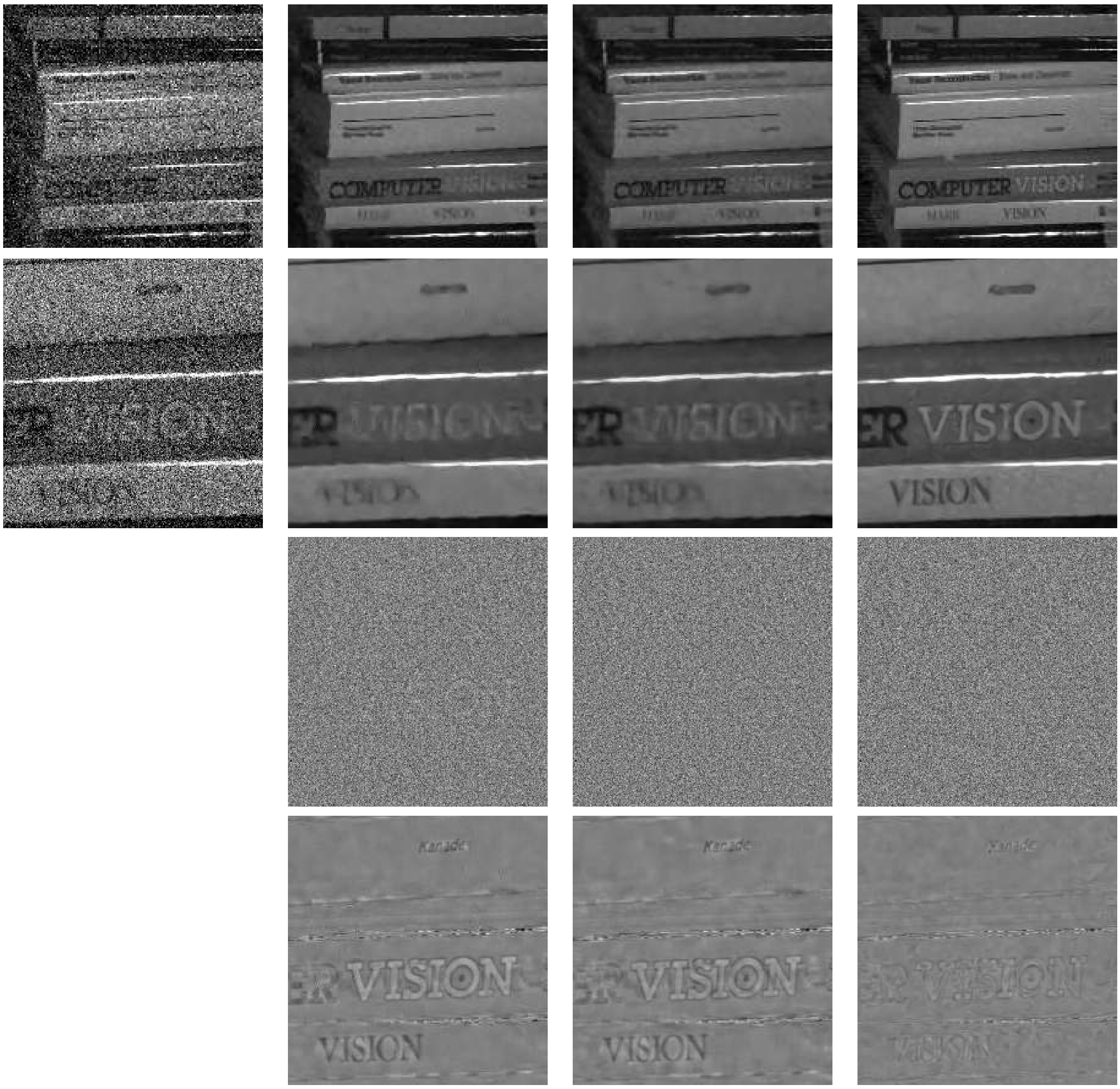


Figure 8: Denoising results for the *iseq* sequence with $\sigma = 50$. First row, from left to right: noisy frame, VBM3D (RMSE=7.63), VBM4D (RMSE=7.33), SPTWO (RMSE=4.92). Second row: detail of the above images. Third row: differences with respect to the noisy frame. Fourth row: differences with respect to the original (noise-free) frame.

5.1 Results on Color Images

Four sequences, composed of 8 frames each, have been used to test the performance of the proposed algorithm on noisy color sequences (see Figure 4).

The results of the compared algorithms, in terms of RMSE, are shown in Table 3. In this case we display the average RMSE value of the three channels. Again, it can be observed that the proposed method outperforms state-of-the-art methods VBM3D (in this case the color version of the algorithm) and VBM4D. Surprisingly, for color sequences the performance of VBM3D is better than that of VBM4D, even for high levels of noise.

	army	cooper	dog	truck	**average**
$\sigma = 10$					
VBM3D	2.96	4.31	4.48	3.76	3.88
VBM4D	3.26	4.48	4.47	4.04	4.06
SPTWO	**2.78**	**4.07**	**4.07**	**3.62**	**3.63**
$\sigma = 20$					
VBM3D	4.42	6.84	6.37	5.70	5.83
VBM4D	5.02	7.15	6.46	6.32	6.24
SPTWO	**3.95**	**6.21**	**5.67**	**5.32**	**5.28**
$\sigma = 30$					
VBM3D	5.54	8.87	7.75	7.30	7.37
VBM4D	6.40	9.25	8.00	8.12	7.94
SPTWO	**4.70**	**7.86**	**6.82**	**6.66**	**6.51**
$\sigma = 40$					
VBM3D	6.40	10.49	8.76	8.63	8.57
VBM4D	7.62	10.99	9.25	9.67	9.38
SPTWO	**5.32**	**9.21**	**7.72**	**7.81**	**7.51**
$\sigma = 50$					
VBM3D	7.34	12.01	9.90	9.92	9.79
VBM4D	8.71	12.49	10.55	11.01	10.69
SPTWO	**5.93**	**10.38**	**8.56**	**8.77**	**8.40**

Table 3: RMSE results for color sequences. The values correspond to the RMSE (averaged over the three channels) computed for the central frame of each sequence. The average RMSE for each method and each noise level is displayed in the last column.

Finally, Figures 9 and 10 show details of the denoising results. As in the case of gray level images, the proposed method preserves better the textures and details of the scene, while VBM3D and VBM4D tend to blur them.

6 Conclusions

We have described a denoising algorithm combining motion estimation and patch-based denoising algorithms. Motion compensation permits the use of spatio-temporal patches for a more robust comparison while the use of PCA for patch denoising preserves texture and details. Implementation details have been provided and the influence of the different parameters in the denoising result has been analyzed. Finally, the results of the proposed method have been compared with the state-of-the-art. The obtained results (both qualitative and quantitative) illustrate the gain in performance of the proposed approach.

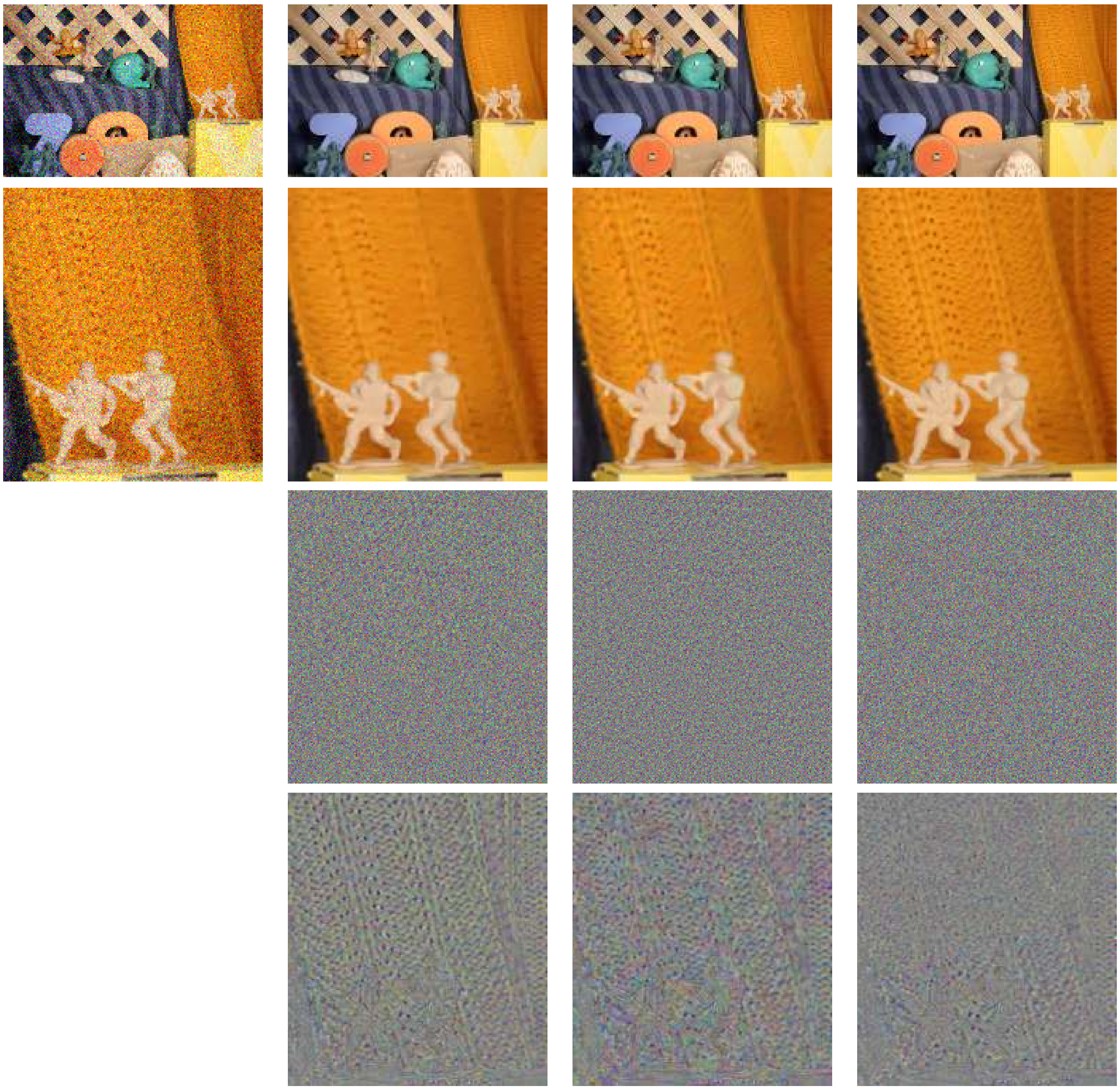


Figure 9: Denoising results for the *army* color sequence with $\sigma = 30$. First row, from left to right: noisy frame, VBM3D (RMSE=5.54), VBM4D (RMSE=6.40), SPTWO (RMSE=4.70). Second row: detail of the above images. Third row: differences with respect to the noisy frame. Fourth row: differences with respect to the original (noise-free) frame.

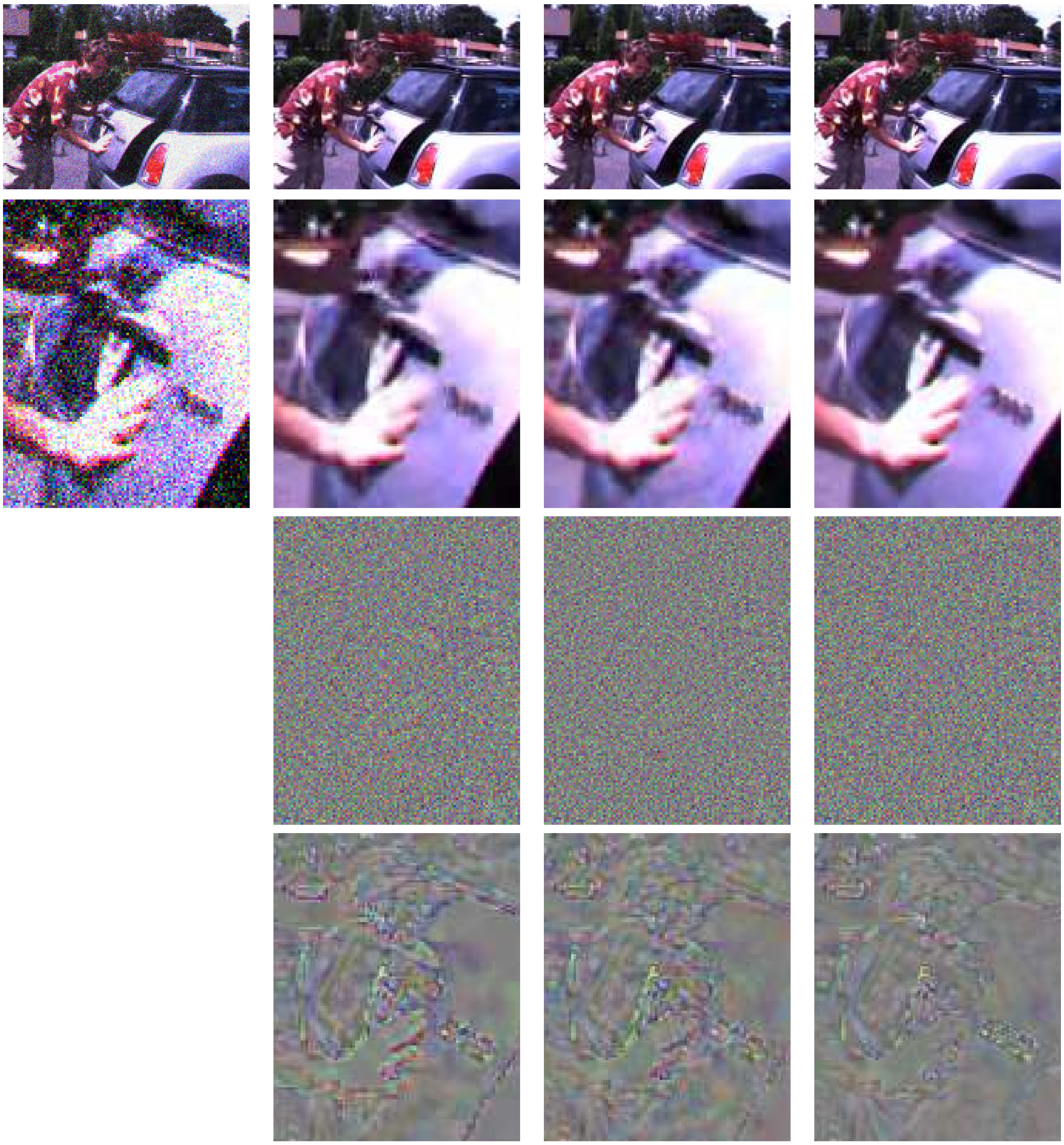


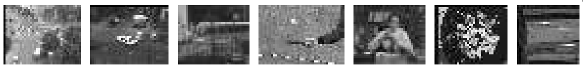
Figure 10: Denoising results for the *cooper* color sequence with $\sigma = 50$. First row, from left to right: noisy frame, VBM3D (RMSE=12.01), VBM4D (RMSE=12.49), SPTWO (RMSE=10.38). Second row: detail of the above images. Third row: differences with respect to the noisy frame. Fourth row: differences with respect to the original (noise-free) frame.

Acknowledgements

The authors were supported by grants TIN2014-53772 and TIN2017-85572-P. The second author was also supported by grants TIN2014-58662-R and DPI2017-86372-C3-3-R.

Image Credits

Middelbury dataset[5].


Standard test sequences.

References

[1] P. Arias and J-M. Morel, *Towards a Bayesian Video Denoising Method*, in International Conference on Advanced Concepts for Intelligent Vision Systems, Springer, 2015, pp. 107–117.

[2] ——, *Video denoising via empirical Bayesian estimation of space-time patches*, Journal of Mathematical Imaging and Vision, (2017). https://doi.org/10.1007/s10851-017-0742-4.

[3] J. Boulanger, C. Kervrann, and P. Bouthemy, *Space-time adaptation for patch-based image sequence restoration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 29 (2007), pp. 1096–1102. https://doi.org/10.1109/TPAMI.2007.1064.

[4] A. Buades, B. Coll, and J.M. Morel, *A non local algorithm for image denoising*, IEEE Computer Vision and Pattern Recognition, 2 (2005), pp. 60–65. https://doi.org/10.1109/CVPR.2005.38.

[5] A. Buades, J.L. Lisani, and M. Miladinović, *Patch Based Video Denoising with Optical Flow Estimation*, IEEE Transactions on Image Processing, 25 (2016), pp. 2573–2586. https://doi.org/10.1109/TIP.2016.2551639.

[6] K. Dabov, A. Foi, and K. Egiazarian, *Video denoising by sparse 3D transform-domain collaborative filtering*, in Proceedings of the 15th European Signal Processing Conference, vol. 1, 2007, p. 7.

[7] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, *Bm3d image denoising with shape-adaptive principal component analysis*, in Proceedings of Workshop on Signal Processing with Adaptive Sparse Structured Representations (SPARS'09), 04 2009.

[8] G. Haro, A. Buades, and J.-M. Morel, *Photographing paintings by image fusion*, SIAM Journal on Imaging Sciences, 5 (2012), pp. 1055–1087. https://doi.org/10.1137/120873923.

[9] H. Ji, S. Huang, Z. Shen, and Y. Xu, *Robust video restoration by joint sparse and low rank matrix approximation*, SIAM Journal on Imaging Sciences, 4 (2011), pp. 1122–1142. https://doi.org/10.1137/100817206.

[10] H. Ji, C. Liu, Z. Shen, and Y. Xu, *Robust video denoising using low rank matrix completion*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2010, pp. 1791–1798. https://doi.org/10.1109/CVPR.2010.5539849.

[5] http://vision.middlebury.edu/flow/data/

[11] Ian Jolliffe, *Principal component analysis*, Wiley Online Library, 2002. ISBN 978-0-387-95442-4.

[12] M. Lebrun, A. Buades, and J.-M. Morel, *A nonlocal Bayesian image denoising algorithm*, SIAM Journal on Imaging Sciences, 6 (2013), pp. 1665–1688. https://doi.org/10.1137/120874989.

[13] H.Y. Lee, W.L. Hoo, and C.S. Chan, *Color video denoising using epitome and sparse coding*, Expert Systems with Applications, 42 (2015), pp. 751–759. https://doi.org/10.1016/j.eswa.2014.08.033.

[14] M. Maggioni, G. Boracchi, A. Foi, and K. Egiazarian, *Video denoising using separable 4D nonlocal spatiotemporal transforms*, in IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, 2011, pp. 787003–787003. https://doi.org/10.1117/12.872569.

[15] J. Mairal, G. Sapiro, and M. Elad, *Learning multiscale sparse representations for image and video restoration*, Multiscale Modeling & Simulation, 7 (2008), pp. 214–241. https://doi.org/10.1137/070697653.

[16] M. K Ozkan, M.I. Sezan, and A.M. Tekalp, *Adaptive motion-compensated filtering of noisy image sequences*, IEEE Transactions on Circuits and Systems for Video Technology, 3 (1993), pp. 277–290. https://doi.org/10.1109/76.257217.

[17] M. Protter and M. Elad, *Image sequence denoising via sparse and redundant representations*, IEEE Transactions on Image Processing, 18 (2009), pp. 27–35. https://doi.org/10.1109/TIP.2008.2008065.

[18] J. Sánchez-Pérez, E. Meinhardt-Llopis, and G. Facciolo, *TV-L1 Optical Flow Estimation*, Image Processing On Line, 3 (2013), pp. 137–150. https://doi.org/10.5201/ipol.2013.26.

[19] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in Sixth International Conference on Computer Vision, 1998, pp. 839–846.

[20] B. Wen, S. Ravishankar, and Y. Bresler, *Video denoising by online 3D sparsifying transform learning*, in 2015 IEEE International Conference on Image Processing (ICIP),, IEEE, 2015, pp. 118–122. https://doi.org/10.1109/ICIP.2015.7350771.

[21] L. Zhang, W. Dong, D. Zhang, and G. Shi, *Two-stage image denoising by principal component analysis with local pixel grouping*, Pattern Recognition, 43 (2010), pp. 1531–1549. https://doi.org/10.1016/j.patcog.2009.09.023.