Compressive Sensed Video Recovery via Iterative Thresholding with Random Transforms

Evgeny Belyaev 1,*, Marian Codreanu 2, Markku Juntti 3, and Karen Egiazarian 4

1 International Laboratory “Computer Technologies”, ITMO University, 197101 Saint-Petersburg, Russia
2 Department of Science and Technology, Linköping University, 581 83 Linköping, Sweden
3 Centre for Wireless Communications, Oulu University, 90014 Oulu, Finland
4 Department of Signal Processing, Tampere University, 33720 Tampere, Finland
* Email: eabelyaev@itmo.ru

Abstract: We consider the problem of compressive sensed video recovery via iterative thresholding algorithm. Traditionally, it is assumed that some fixed sparsifying transform is applied at each iteration of the algorithm. In order to improve the recovery performance, at each iteration the thresholding could be applied for different transforms in order to obtain several estimates for each pixel. Then the resulting pixel value is computed based on obtained estimates using simple averaging. However, calculation of the estimates leads to significant increase in reconstruction complexity. Therefore, we propose a heuristic approach, where at each iteration only one transform is randomly selected from some set of transforms. First, we present a simple example, when block-based 2-D discrete cosine transform is used as the sparsifying transform, and show that the random selection of the block size at each iteration significantly outperforms the case when fixed block size is used. Then, we show that similar improvement can be achieved, when a random shift of the blocks grid is introduced, or when random 2-D discrete wavelet transform is applied at each iteration. Second, building on these simple examples, we apply the proposed approach when video block-matching and 3D filtering (VBM3D) is used for the thresholding and show that the recovery performance could be improved as compared with the recovery based on VBM3D with fixed transform. Finally, extending the introduced approach, we propose to also randomly select a frame residual computation algorithm as well and show that it could provide additional gain as well.

1 Introduction

During the last decade, compressive sensing (CS) [1, 2] became a very promising theory which can be used for development of new power-efficient sensors for signals of different nature. In the case of video sequences, a small number (e.g., 5-20%) of random linear measurements are observed for each frame. These measurements are computed by processing raw pixel data of a conventional sensor or even directly coming out from a modified sensor in which the processing circuitry is integrated as a part of the sensing circuitry itself and realized in the analog domain [3, 4]. A linear reconstruction by inverse transform, cannot, in general, recover the signal from a small number of measurements. However, in [1, 2], it was shown that if the signal is sparse in some known transform domain (e.g., wavelet), then a stable reconstruction is possible. In case of video sequences, CS sampling could be done in a spatiotemporal manner. However, this could be impractical to implement in a real-life sensing device [5]. Therefore, it is more realistic to consider frame-by-frame sensing models which could be realized for a whole frame (global sensing) or for each non-overlapped block within a frame (block-based sensing).

In a global sensing model including combination of low-pass DCT and noiselet coefficients is considered. This sensing model could be performed without matrix multiplication and allows low-resolution preview of a sensed frame without any computations. For this model, it was demonstrated that an image can be reconstructed from compressive measurements via mathematical programming, where the sparsest representation in the transform domain that gives the observed measurements is searched [6]. Herewith, \( l_1 \) norm is used as the sparsity criterion. Later, in [7], this work has been extended to video sequences, where sparsity of motion compensated frame differences are exploited as well. Taking into account that motion information is not available at the decoder side, this approach requires several iterations of motion refinement, so that at the current iteration the motion information is estimated from the video sequence recovered at the previous iteration.

The block-based sensing could be applied separately for each block, i.e., it can be done in parallel. It requires much less memory needed for storage of sensing matrix and enables fast reconstruction via separate recovery of each block [8–10]. In [11–13], multihypothesis predictions of the current frame are generated from previously reconstructed frames. The predictions are used to obtain a residual in the measurements domain and regularization is performed assuming that the residual is typically more compressible than the original signal. In [5] this approach was extended introducing the reweighted residual sparsity model, where the sparsity of different residual coefficients is discriminatively weighted and the weights are iteratively updated.

In this paper we consider a signal recovery based on iterative thresholding in some sparse transform domain [14–16]. In [17, 18] it was shown that Block-Matching and 3D filtering (BM3D) algorithm [19] can be used as thresholding operator for an image. BM3D allows to achieve a high quality of reconstruction, because the transform embedded in BM3D achieves relatively high sparsity level of representation by exploiting both local similarity of neighbor pixels and non-local self-similarity of images. In [20, 21], this approach has been extended to compressive sensed video sequences recovery using an extension of BM3D called Video Block-Matching and 3D filtering (VBM3D) [22]. VBM3D can achieve even higher sparsity level of representation for a video sequence, because it also takes into account temporal similarity between blocks in different frames.

The mentioned above recovery techniques based on iterative thresholding (such as [17, 18, 20, 21]) assume that there exist a single sparsifying transform which provides the highest sparsity level among others candidates. Utilizing this transform guarantees
the maximum reconstruction performance. The same idea underlies other thresholding free recovery approaches (such as [5–7]). However, the assumption is not always valid for images and video sequences having heterogeneous structures. For example, a relatively small block size of a transform could help to achieve a high sparsity level on image areas with textures, while it could be more efficient to apply larger block size on flat areas. One solution to take this phenomenon into account is to recover a signal several times utilizing different sparsifying transforms, obtain several estimates for each pixel, and then compute the resulting pixel value from the estimates using simple averaging (see our previous work [23]). However, such an approach leads to significant increase of computational complexity needed for the recovery. In order to overcome this limitation, instead of averaging, at each iteration we propose to randomly select the sparsifying transform from some set of transforms. We show that the random selection achieves better reconstruction quality compared with a case when only a single transform is used without significant increase of computational complexity.

The main contributions of this paper are the following:

1. As an illustrative example, we consider an iterative soft thresholding algorithm based on well known block-based 2-D discrete cosine transform (DCT). First, we define a set of five sparsifying transforms as \{4 × 4 DCT, 8 × 8 DCT,..., 64 × 64 DCT\}, apply the thresholding for each transform in order to get five estimates for each pixel and compute the resulting pixel value using simple averaging. We show that this approach outperforms the recovery, when only one transform from the set is applied for all iterations with a price of computational complexity. At each iteration we randomly select only one transform from the set and show that the resulting recovery performance is close to the approach utilizing the averaging. Finally, we show that similar results can be achieved, when a random shift of the blocks grid is utilized, or when random 2-D discrete wavelet transform is applied at each iteration.

2. Following the same reasoning, we extend the proposed approach, when VBM3D is used as the thresholding operator. In this case the introduced set contains 576 sparsifying transforms, i.e., the averaging cannot be applied due to extremely high computational complexity. We show that the random selection of the sparsifying transform embedded in VBM3D provides up to 0.54 dB improvement in Peak Signal-to-Noise Ratio (PSNR) as compared with the case where the transform is fixed as it is defined in its "High Profile". In comparison to the state-of-the-art algorithms from [7], [20] and [5], it provides up to 6.19, 4.5 and 3.12 dB improvement in PSNR, respectively.

3. Extending the introduced approach, we propose to randomly select a frame residual computation algorithm as well. As an example, we randomly use simple back projection or total variation minimization, and show that it provides additional gain up to 1.24 dB.

The rest of the paper is organized as follows. In Section 2 we describe the considered sensing models. Section 3 is dedicated to the proposed recovery via iterative soft thresholding. Section 4 introduces the proposed soft thresholding with random transforms for the simple 2-D DCT and 2-D DWT cases and for more complex VBM3D case. The experimental results showing the advantages of the proposed recovery algorithm are presented in Section 5. Conclusions are drawn in Section 6.

In this paper, we use the following notations. The column vectors and matrices are denoted by boldfaced lowercase and uppercase letters, respectively, e.g., \(\mathbf{v}\) and \(\mathbf{A}\). The superscript \((\cdot)^T\) denotes the transpose operation for a vector or a matrix, \((\cdot)^{-1}\) is inverse operation, \(\|\cdot\|_p\) denotes \(p\)-norm of a vector, \(\|\cdot\|_F\) denotes the Euclidean norm, \(\otimes\) denotes the Kronecker product, vec(\(\mathbf{A}\)) concatenates columns of \(\mathbf{A}\) into a vector, \([\mathbf{v}_1; \mathbf{v}_2]\) concatenates vectors \(\mathbf{v}_1\) and \(\mathbf{v}_2\) into a vector, \(\hat{\mathbf{v}}\) means estimation of a vector \(\mathbf{v}\), \(\leftarrow\) means assignment operation, \(\lfloor \cdot \rfloor\) denotes a number of elements in vector or matrix.

## 2 Sensing Models

### 2.1 Sensing Model via Noiselet Transform

Consider a video sequence \(X_1, \ldots, X_F\), where each frame \(X_i \in \mathbb{R}^{N \times N}\) is 2-D signal (image) of size \(N \times N\) pixels. The linear measurements are acquired for each frame independently on the other frames as

\[
y_i = \mathbf{\Phi}_i x_i,
\]

where \(x_i = \text{vec}(X_i), \mathbf{\Phi}_i \in \mathbb{R}^{M_i \times N^2}, M_i < N^2\), denotes the measurement matrix for \(i\)th frame. The ratio \(\frac{M_i}{N^2}\) is called the sensing rate for frame \(i\).

Without loss of generality, we define \(\mathbf{\Phi}_i\) as

\[
\mathbf{\Phi}_i = \begin{bmatrix} \mathbf{W} \\ \mathbf{N}_i \end{bmatrix},
\]

where \(\mathbf{W}\) is downsampling matrix of the Haar wavelet transform which is needed to compute low-resolution image size of \(\frac{N}{N_0} \times \frac{N}{N_0}\) and \(\mathbf{N}_i\) is \(M_i \times \frac{N}{N_0} \times \frac{N}{N_0}\) random rows of orthogonal real-valued dragon noiselet transform matrix [6] (see Fig. 1, a). We assume that \(N\) is divisible by \(2^k\). Thus, measurements \(y_i\) include two parts

\[
y_i = \begin{bmatrix} \mathbf{W} \\ \mathbf{N}_i \end{bmatrix} x_i = \begin{bmatrix} \mathbf{W} \mathbf{x}_i \\ \mathbf{N}_i \mathbf{x}_i \end{bmatrix} = \begin{bmatrix} \mathbf{y}_i^W \\ \mathbf{y}_i^N \end{bmatrix},
\]

where \(\mathbf{y}_i^W\) is low-resolution image and \(\mathbf{y}_i^N\) is noiselet coefficients. The measurement matrix (2) has the following advantages. First, operations such as \(y_i = \mathbf{\Phi}_i x_i\) and \(\hat{x}_i = \mathbf{\Phi}_i^T y_i\) can be performed without matrix multiplication. Second, the part of measurements \(\mathbf{y}_i^N\) allows low-resolution preview of the video sequence without any computations. Please see more details related to the measurement matrix implementation in Appendix A.

Finally, we utilize different number of measurements for each frame: a video sequence is divided into non-overlapped groups of \(G\) frames, so that the first frame has \(\alpha\) times more measurements than any remaining frame within the group, i.e., \(M_1 = \alpha M_2 = \ldots = \alpha M_G = M_G+1\) and so on. Frames with more measurements have the same sense as a key frames in conventional hybrid video coding algorithms: they help to recover frames with less measurements [7].

### 2.2 Block-based Sensing Model

Following [8] we also consider block-based random measurements applied frame by frame. Each video frame \(X_i\) is divided into nonoverlapping blocks of size \(B \times B\). Then each block is acquired utilizing a measurement matrix \(\Omega_B\) of size \(M_B \times B^2\). As a result, the sensing matrix for each frame is the same and equal to

\[
\mathbf{\Phi}_i = \begin{bmatrix} \Omega_B & 0 & 0 & 0 \\ 0 & \Omega_B & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \Omega_B \end{bmatrix}.
\]

Comparing to the noiselet based sensing, the block-based sensing could be applied separately for each block, i.e., it can be done in parallel. Moreover, it requires much less memory needed for storage of sensing matrix, since we need to store only \(\Omega_B\) only for one block instead of set \(\mathbf{\Phi}_1, \ldots, \mathbf{\Phi}_F\). Following [5], in our experiments \(\Omega_B\) corresponds to a Gaussian random projection matrix for block size of \(32 \times 32\), i.e., \(B = 32\).

## 3 Recovery via Iterative Soft Thresholding

### 3.1 Reconstruction via Iterative Soft Thresholding

For the reconstruction of a video sequence \(X_1, \ldots, X_F\) from compressive measurements \(y_1, \ldots, y_F\), we use an iterative soft thresholding algorithm [15, 24], where the frames at iteration \(k\) are estimated
as:

\[ \{x^k_1, \ldots, x^k_F\} = \text{soft} \left( \{x_1^{k-1} + \beta^k_i \Delta x_i^k, \ldots, x_F^{k-1} + \beta^k_F \Delta x_F^k\}, \sigma_i^k \right), \]  

where \( \beta^k_i \) is step-size, \( 0 < \beta^k_i < \frac{2}{\| \Phi \|^2} \) [24], and \( \Delta x_i^k \) is a residual defined as

\[ \Delta x_i^k = \Phi_i^T (y_i - \Phi_i \hat{x}_i^{k-1}). \]  

(H5)

Initial estimates \( \hat{x}_1^0, \ldots, \hat{x}_F^0 \) are zero-vectors.

Soft thresholding operator \( \text{soft}(\{x_1, \ldots, x_F\}, \sigma) \) includes three main steps:

1. A sparsifying transform with matrix \( \Psi \) is applied for a video sequence as \( \hat{\theta} = \Psi \{x_1; \ldots, x_F\} \).
2. Soft thresholded transform coefficients \( \hat{\theta}_i = \{\hat{\theta}_i\} \) are calculated as

\[ \hat{\theta}_i = \begin{cases} 0, & \text{if } |\theta_i| < \sigma, \\ 1 - \frac{\sigma}{|\theta_i|} \theta_i, & \text{otherwise}, \end{cases} \]  

(7)

3. A soft thresholded video sequence is calculated as \( [\hat{x}_1; \ldots; \hat{x}_F] = \Psi^{-1} \hat{\theta} \).

An optimal selection of \( \sigma_k \) and \( \beta^k_i \) at each iteration \( k \) and the number of iterations \( K \) for a given measurement matrix and sparsifying transform is an open research problem. In this paper we keep the step-size constant, i.e., \( \beta^k_i = \beta \), and reduce \( \sigma_k \) from iteration to iteration using quadratic function [18], i.e.,

\[ \sigma_k = \sigma_0 \left( \frac{K - k + 1}{K} \right)^2, \]  

(8)

where \( \sigma_0 \) is a starting threshold value. For each sensing rate we jointly select parameters \( \sigma_0 \) and \( K \) which provide the maximum reconstruction quality for a test video sequence and then use them for the remaining video sequences.

Thus, combining (5), (6) and (8), we reconstruct the video sequence from the compressive measurements as it is shown in Algorithm 1.

![Algorithm 1](image)

**Algorithm 1**: Recovery via iterative soft thresholding

**Input**: \( y_1, \ldots, y_F, K, \sigma_0 \)

1. for \( i = 1, \ldots, F \) do
2. \( x_i^0 \leftarrow 0 \)
3. end for

4. for all \( k = 1, \ldots, K \) do
5. if \( k \neq K \) then
6. for all \( i = 1, \ldots, F \) do
7. \( \Delta y_i^k \leftarrow y_i - \Phi_i x_i^{k-1} \)
8. \( \Delta x_i^k \leftarrow \Phi_i^T \Delta y_i^k \)
9. \( \hat{x}_i^k \leftarrow \hat{x}_i^{k-1} + \beta \Delta x_i^k \)
10. end for
11. \( \sigma_k \leftarrow \sigma_0 \left( \frac{K - k + 1}{K} \right)^2 \)
12. \( [\hat{x}_1^k; \ldots; \hat{x}_F^k] \leftarrow \text{soft}(\{\hat{x}_1^k; \ldots; \hat{x}_F^k\}, \sigma_k) \)
13. else
14. for all \( i = 1, \ldots, F \) do
15. \( \Delta y_i^k \leftarrow y_i - \Phi_i x_i^{k-1} \)
16. \( \Delta x_i^k \leftarrow \Phi_i^T \Delta y_i^k \)
17. \( \hat{x}_i^k \leftarrow \hat{x}_i^{k-1} + \Delta x_i^k \)
18. end for
19. end if
20. end for

between observed measurements \( y_i \) and projection \( \Phi_i x_i^{k-1} \) of the frame estimated at iteration \( k-1 \) into the measurement domain is calculated (see step 7). Then the residual \( \Delta y_i^k \) is projected back into an image domain to calculate a residual frame \( \Delta x_i^k \) (see step 8).

![Fig. 1](image)

**Fig. 1**: Illustration of a) Sensing model for frame \( x_i \) utilizing (3), b) Implementation of \( x_i = \hat{\Phi}_i^T y_i \).
3.2 Implementation issues

There is one issue in practical implementation of Algorithm 1, when the noiselet matrix model considered in Section 2.1 is utilized. The value $\|\Phi_i\|^2_F$, which bounds the maximum possible value for $\beta$, varies depending on a sensing rate and the rows indexes randomly selected from the noiselet matrix (see Fig. 2). It gives difficulties in $\beta$ selection, because at the recovery side we need to calculate $\|\Phi_i\|^2_F$ value for each frame.

To avoid it, we calculate the residual frame $\Delta x_i$ from the residual measurements $\Delta y_i = [\Delta y_i^n; \Delta y_i^w]$ as

$$\Delta x_i = W^T \Delta y_i^w + N_i^T \Delta y_i^n - W^T \bar{\Delta} y_i^w,$$

(9)

where $\bar{\Delta} y_i^w = WN_i^T \Delta y_i^n$ is an estimation of low-resolution residual image $\Delta y_i^n$ obtained from incomplete residual noiselet measurements $\Delta y_i^n$, i.e., $\bar{\Delta} y_i^w$ is a redundant component which can be removed without any loss of information. The residual frame calculation (9) is equivalent to the sensing via matrix

$$\bar{\Phi}_i = \begin{bmatrix} W & N_i - N_iW^T W \end{bmatrix}.$$  

(10)

The main advantage of the equivalent matrix $\bar{\Phi}_i$ is $\|\bar{\Phi}_i\|^2_F = 1$ (see Appendix 2), i.e., $0 < \beta < 2$ (see Fig. 2). At the same time, we do not need to change the sensing model, i.e., we perform the sensing via $\Phi$, but the residual frame in Algorithm 1 is calculated as $\Delta x_i^n = \bar{\Phi}_i^T \Delta y_i^n$.

Fig. 1, b) shows the implementation of the multiplication by matrix $\bar{\Phi}_i^T$. First, we reconstruct the residual using only noiselet coefficients residual $\Delta x_i^n$, as

$$\Delta x_i^n = N_i^T \Delta y_i^n.$$  

(11)

After that, we apply the forward Haar wavelet transform for $\Delta x_i^n$, as

$$\begin{bmatrix} \Delta \bar{y}_i^w \\ h_i \end{bmatrix} = H \Delta x_i^n,$$

(12)

where $H$ is the Haar transform matrix, $h_i$ is a vector of wavelet coefficients except the coefficients corresponding to the low-resolution image. Then we insert $\Delta \bar{y}_i^w$ instead of $\Delta \bar{y}_i^w$, and perform the inverse Haar wavelet transform, so that

$$\Delta x_i = H^T \begin{bmatrix} \Delta \bar{y}_i^w \\ h_i \end{bmatrix}.$$  

(13)

Matrix multiplications in (11), (12) and (13) can be implemented utilizing fast functions, i.e., there is no necessity to use a high dimensional matrices to calculate the residual $\Delta x_i$.

3.3 Objective visual quality evaluation

For objective visual quality evaluation of the recovery algorithm we use Structural Similarity Index (SSIM) [25], which measures the similarity between two images and ranges from 0 to 1, and Peak Signal-to-Noise Ratio (PSNR) defined as

$$\text{PSNR} = 10 \log_{10} \left( \frac{x_{\text{max}}^2}{d} \right),$$

(14)

where $x_{\text{max}} = 255$ is maximum possible value of pixel in original video sequence, and

$$d = \frac{1}{N^2F} \sum_{f=1}^{F} \sum_{i=1}^{N} \sum_{j=1}^{N} (\hat{X}(i,j,f) - \tilde{X}(i,j,f))^2,$$

(15)

where $X(i,j,f)$ and $\tilde{X}(i,j,f)$ are values of pixels with coordinates $(i,j)$ in original and reconstructed frames with number $f$, respectively.

4 Recovery via Iterative Soft Thresholding with Random Transforms

4.1 Recovery via 2-D DCT and 2-D DWT

The recovery performance of Algorithm 1 highly depends on transform matrix $\Psi$ used in the soft thresholding operator, i.e., higher sparsity level of a signal representation achieved by $\Psi$ means better quality of reconstruction. Let us assume that only one frame (or image) should be recovered, i.e., $F = 1$. Then, in a simple case, 2-D DCT can be used to obtain moderate sparsity level exploiting local similarity of neighbor pixels within an image. In this case, at each iteration, an input frame $X$ is divided into non overlapped blocks size of $L \times L$. Let us denote the $(i,j)$-th block of $X$ as $X_{i,j} \in \mathbb{R}^{L \times L}$, and process each block in a raster scan, i.e., from left to right and from top to bottom: $X_{0,0}$, $X_{0,1}$, $X_{0,2}$, ..., $X_{1,0}$, $X_{1,1}$ and so on. Then, step 12 of Algorithm 1 is realized in the following way. First, the transform coefficients $\theta_{L \times L}$ are calculated as:

$$\theta_{L \times L} = \Psi_{L \times L}^{-1} \begin{bmatrix} \vec{x}(X_{0,0}) \\ \vec{x}(X_{0,1}) \\ \vdots \\ \vec{x}(X_{1,0}) \end{bmatrix},$$

(16)

where

$$\Psi_{L \times L} = \begin{bmatrix} T_{\text{det}} \otimes T_{\text{det}} \\ T_{\text{det}} \otimes T_{\text{det}} \\ \vdots \\ T_{\text{det}} \otimes T_{\text{det}} \end{bmatrix},$$

(17)

$T_{\text{det}}$ is $L \times L$ matrix of 1-D DCT (type II) for column-vector size of $L$. Then, the soft thresholded frame at iteration $k$ is calculated as:

$$\Psi_{L \times L}^{-1} \begin{bmatrix} T_{\text{det}}^T \otimes T_{\text{det}}^T \\ T_{\text{det}}^T \otimes T_{\text{det}}^T \end{bmatrix} \begin{bmatrix} \vec{x}(X_{0,0}) \\ \vec{x}(X_{0,1}) \\ \vdots \\ \vec{x}(X_{1,0}) \end{bmatrix}.$$  

(18)

where $\theta_{L \times L}$ is calculated using $\theta_{L \times L}$ and threshold $\sigma_{k_i}$ as in (7), and

$$\Psi_{L \times L}^{-1} = \begin{bmatrix} T_{\text{det}}^T \otimes T_{\text{det}}^T \\ T_{\text{det}}^T \otimes T_{\text{det}}^T \end{bmatrix},$$

(19)
Finally, blocks $\tilde{X}_{0,0}, \tilde{X}_{0,1}, ..., \tilde{X}_{1,0}, ...$ form the soft thresholded frame $\tilde{X}$.

Sparsity level achieved by 2-D DCT depends on an image properties: large block sizes are more efficient for a flat areas, while small ones are better for areas with many details. Let us define a set of possible $L$ values as $\mathcal{L}$. Then we can choose the block size $L \times L, L \in \mathcal{L}$ to achieve some balance in performance for flat and for detailed areas. However, the great variety in images properties makes difficult for any fixed size 2-D DCT to achieve good sparsity level for all possible cases. To overcome this disadvantage, we could use an approach proposed in our previous work [23] in the following way. At each iteration $k$, we can apply the soft thresholding for each $L \in \mathcal{L}$ to obtain $\tilde{E} = |\mathcal{L}|$ estimates for each pixel. Then we can calculate resulting pixel value based on obtained estimates using simple averaging, which is well-known in signal denoising [19], i.e.,

$$
\begin{bmatrix}
    \text{vec}(\tilde{X}_{0,0}) \\
    \text{vec}(\tilde{X}_{0,1}) \\
    \text{vec}(\tilde{X}_{1,0}) \\
    \vdots
\end{bmatrix} = \frac{1}{\tilde{E}} \sum_{L \in \mathcal{L}} \Psi^{-1}_{L \times L} \tilde{E}_{L \times L}.
$$

However, calculation of the estimates for each block size leads to significant increase in reconstruction complexity. Therefore, we propose to use at each iteration of Algorithm 1 a random value for $L$ chosen with uniform probabilities from the set $\mathcal{L}$, and then to apply the soft thresholding for the selected block size $L \times L$ using (18). Figs. 3–5 illustrate the performance of the proposed method for images size of $128 \times 128$, when the block size can be chosen from set $\{4 \times 4, 8 \times 8, 16 \times 16, 32 \times 32, 64 \times 64\}$, i.e., $\mathcal{L} = \{4, 8, 16, 32, 64\}$, $\tilde{E} = 5$, and for $\sigma_0 = 70, K = 100$ and $\beta = 1.75$. One can see that the soft thresholding based on fixed block size gives very low reconstruction quality, while both the averaging (20) and random block size selection provides much better performance. Note that, in the case of random block size selection we need to calculate the forward and inverse 2-D DCT only ones at each iteration, while the averaging method (20) requires five corresponding operations.

Let us consider two more examples. In the first example we use $8 \times 8$ DCT as a sparsifying transform with shift of blocks grid by random vector $(s_x, s_y), s_x \in \{-1, 0, 1\}, s_y \in \{-1, 0, 1\}$ as it is illustrated in Fig. 9, i.e., $\tilde{E} = 9$. In the second example we randomly select a sparsifying transform from set of 2-D DWT transforms denoted in MATLAB as ‘Haar’, ‘coif5’, ‘sym8’, ‘bior4.4’ and ‘db45’, i.e., $\tilde{E} = 5$. Figs. 6–8 and 10–12 show the corresponding performance. One can see, that the performance of the proposed soft thresholding with random transform is much better than the soft thresholding with any fixed transform and similar to the performance
of the soft thresholding with averaging. At the same time, the computational complexity of the proposed approach is much lower than in the case of averaging, because only one transform instead of $\mathcal{E}$ is used at each iteration of Algorithm 1. Table 1 shows execution time needed for recovery of 5 frames size of $128 \times 128$ via techniques mentioned above implemented in MATLAB. Here we define $T(\Psi_i)$ as an execution time for the iterative thresholding algorithm based on sparsifying transform $\Psi_i$. It could be noticed that the proposed random transform selection is from 3 to 6 times faster than the averaging.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Scenario & $\min_{i=1}^5 T(\Psi_i)$ & $\max_{i=1}^5 T(\Psi_i)$ & Averaging & Random \\
\hline
2-D DCT & 13 & 86 & 103 & 29 \\
Grid shift & 47 & 51 & 306 & 48 \\
2-D DWT & 15 & 84 & 96 & 27 \\
\hline
\end{tabular}
\end{table}

### 4.2 Recovery via VBM3D

Considered above 2-D DCT and 2-D DWT achieve some moderate sparsity level of representation using local redundancy of images, however for higher sparsity level of representation, non-local self-similarity of images should be exploited as well. In [19], an image denoising algorithm exploiting both local and non-local image similarities has been proposed. In general, BM3D, first, achieves a highly sparse representation of an image using transform, which depends on an image, and then, performs hard thresholding and empirical Wiener shrinkage of transform coefficients depending on the noise level. Therefore, as it was shown in [18], BM3D can be also used as a thresholding operator in other applications, e.g., for image super-resolution or compressive sensed image recovery.
VBM3D assumes that a video sequence has temporal similarity between blocks in different frames and similarity between blocks at different spatial locations within the same frame. Under this assumption, at Stage 1 for each reference block, similar operations are performed, but the grouping weights are inversely proportional to the norm of the thresholded 3D values, these estimates are aggregated by weighted averaging, where the final estimate of the pixel values is obtained. In order to get the final estimate of the pixel values, these estimates are aggregated by weighted averaging. Where the weights are inversely proportional to the norm of the thresholded 3D block.

For sparse representation of video sequences, temporal redundancy should be exploited as well. It can be achieved using an extension of BM3D called VBM3D [22]. VBM3D consists of two main stages shown in Algorithm 2 and Algorithm 3, respectively. VBM3D assumes that a video sequence has temporal similarity between blocks in different frames and similarity between blocks at different spatial locations within the same frame. Under this assumption, at Stage 1 for each reference block \( r \) several most similar blocks \( b_1, \ldots, b_K \) size of \( L_H \times L_H \) in the current frame and neighbor frames \( x_{f-\Delta f}, \ldots, x_{f+\Delta f} \) are selected by function \( \text{find()} \) (see line 4). Then these blocks are grouped into a 3D block and hard thresholded using transform \( \Psi_{3D} \) and threshold \( \lambda_{3D}\sigma \). Here, the hard thresholding operator \( \text{hard}[\{b_1; \ldots; b_K\}, \Psi, \sigma] \) (see line 5) includes the following three steps:

1. A sparsifying transform with matrix \( \Psi \) is applied for a 3D block as \( \hat{\theta} = \Psi_{3D} b \).
2. Hard thresholded transform coefficients \( \hat{\theta} = \{\hat{\theta}_i\} \) are calculated as
   \[
   \hat{\theta}_i = \begin{cases} 
   0, & \text{if } |\theta_i| < \sigma, \\
   \theta_i, & \text{otherwise}.
   \end{cases} 
   \] (21)
3. A hard thresholded block is calculated as \( b_1; \ldots; b_K = \Psi^{-1}\hat{\theta} \).

Moreover, operator \( \text{hard()} \) computes a weight coefficient \( \gamma_H = \frac{1}{\sigma^2\|\theta\|_0^2} \), where \( \|\theta\|_0 \) is \( l_0 \)-norm (number of non-zero coefficients) of \( \theta \). This weight is utilized to get the final estimate of each pixel value via weighted averaging in the following way. First, a frame index \( t \) and spatial coordinates \((i, j)\) determining location of block \( b_0 \) in frame \( x_t \) are extracted via function \( \text{getcoord()} \) (see line 7). Second, operator \( \text{insert()} \) (see line 8) creates two zero-frames \( s \) and \( e \).
Then it inserts block \( \tilde{b}_k \) into frame \( s \) and a block of the same size containing only ones into frame \( e \) using coordinates \((i, j)\) for both blocks. Finally, a sum of weighted pixels values and corresponding sum of weights are accumulated in buffers \( x_1, ..., x_F \) and \( w_1, ..., w_F \), respectively, and the resulting output frames \( x_1^H, ..., x_F^H \) are computed in line 15, where the division is performed in an element-by-element manner.

Algorithm 2: VBM3D video denoising algorithm (Stage 1)

Input: \( x_1, ..., x_F, \sigma \)

1. \( x_k \leftarrow 0, ..., x_F \leftarrow 0, w_1 \leftarrow 0, ..., w_F \leftarrow 0 \)
2. for \( f = 1, ..., F \) do
3. for each reference block \( r \in x_f \) size of \( L_H \times L_H \) do
4. \{\( b_1, ..., b_K \)\} \( \leftarrow \text{find}(r, x_f - \Delta_f, ..., x_f + \Delta_f) \)
5. \{\( b_1, ..., b_K, \gamma_H \} \leftarrow \text{hard}(\{\|b_1; ..., b_K\|; \Psi_{HD}, \lambda_{HD}\sigma\}
6. for \( k = 1, ..., K \) do
7. \( \{i, j, t\} \leftarrow \text{getcoord}(b_k) \)
8. \( \hat{x}_i \leftarrow \hat{x}_i + \gamma_H s \)
9. \( w_i \leftarrow w_i + \gamma_H e \)
10. end for
11. end for
12. end for
13. end for
14. for \( f = 1, ..., F \) do
15. \( \tilde{x}_f^H \leftarrow \hat{x}_f/w_f \)
16. end for
17. end for

The resulting frames \( \tilde{x}_1^H, ..., \tilde{x}_F^H \) are used at Stage 2 as an input (see Algorithm 3). Here, instead of the hard thresholding, an empirical Wiener shrinkage operator \( \text{wiener}((\{b_1; ..., b_K\}; \Psi, \sigma) \) (see line 5) is applied. It includes the following three steps:

1. A sparsifying transform with matrix \( \Psi \) is applied for a 3D block as \( \theta = \Psi [b_1; ..., b_K] \).
2. A shrinkage coefficients \( \tilde{\theta} = (\tilde{\theta}_i) \) are calculated as \( \tilde{\theta}_i = \theta_i/\mu_i \), where \( \mu_i = \sqrt{\sigma^2 + \tilde{\theta}_i^2} \).
3. A resulting block is calculated as \( \{b_1; ..., b_K\} = \Psi^{-1} \tilde{\theta} \).

Moreover, operator \( \text{wiener}() \) computes a weight coefficient \( \gamma_W \) for all pixels estimates corresponding to the 3D block as \( \gamma_W = \frac{1}{\sqrt{\mu^2}} \). All other processing is similar to Stage 1 of the algorithm. One can see that VBM3D could be considered as a thresholding operator, i.e., it can be used for compressive sensed video recovery [20, 21].

Algorithm 3: VBM3D video denoising algorithm (Stage 2)

Input: \( \tilde{x}_1^H, ..., \tilde{x}_F^H, \sigma \)

1. \( x_k \leftarrow 0, ..., x_F \leftarrow 0, w_1 \leftarrow 0, ..., w_F \leftarrow 0 \)
2. for \( f = 1, ..., F \) do
3. for each reference block \( r \in \tilde{x}_f^H \) size of \( L_W \times L_W \) do
4. \{\( b_1, ..., b_K \)\} \( \leftarrow \text{find}(r, \tilde{x}_f^H - \Delta_f, ..., \tilde{x}_f^H + \Delta_f) \)
5. \{\( b_1, ..., b_K, \gamma_W \} \leftarrow \text{wiener}(\{b_1; ..., b_K\}; \Psi_{WD}, \sigma) \)
6. for \( k = 1, ..., K \) do
7. \( \{i, j, t\} \leftarrow \text{getcoord}(b_k) \)
8. \( \hat{x}_i \leftarrow \hat{x}_i + \gamma_W s \)
9. \( w_i \leftarrow w_i + \gamma_W e \)
10. end for
11. end for
12. end for
13. end for
14. for \( f = 1, ..., F \) do
15. \( \hat{x}_f \leftarrow \hat{x}_f/w_f \)
16. end for

Table 2: \( K \) and \( \sigma_0 \) used in experiments

<table>
<thead>
<tr>
<th>Sensing rate</th>
<th>( \sigma_0 )</th>
<th>( K )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>24</td>
<td>200</td>
</tr>
<tr>
<td>1/10</td>
<td>13</td>
<td>200</td>
</tr>
<tr>
<td>1/20</td>
<td>13</td>
<td>300</td>
</tr>
<tr>
<td>1/50</td>
<td>9</td>
<td>300</td>
</tr>
</tbody>
</table>

Let us denote CS-VBM3D as Algorithm 1 when the thresholding is performed by VBM3D with fixed transform defined in its "High Profile". The parameters \( \sigma_0 \) and \( K \) have been selected experimentally for each sensing rate value. Fig. 13 shows an example of the parameters selection process for video sequence "Foreman" at sensing rate 1/20 for the first 17 frames. One can see that \( K = 300 \) iterations is enough for the reconstruction, because further increase of \( K \) does not give any improvements. At the same time, \( \sigma_0 = 10 \) provides the best reconstruction quality. Using this approach we selected the parameters for each sensing rate (see Table 2). The step-size \( \beta \) was set to 1.75.

Let us denote Proposed 1 as recovery algorithm with random sparsifying transform selection within VBM3D. As an example, we use the following transform parameters of VBM3D to be randomly changed at the both stages of the algorithm:

- Spatial transforms \( \Psi_{HD} \) and \( \Psi_{WD} \) from set {no transform, 2-D DCT, 2-D DWT denoted as 'bior1.5' in MATLAB};
- The spatial transforms sizes \( L_H \times L_H \) and \( L_W \times L_W \) from set \{4 x 4, 8 x 8, 16 x 16, 32 x 32\};
- Sliding step to process every next reference block from set \{L_H \times L_H / 2, L_W / 2, L_W / 2 + 1\} for Stage 1 and Stage 2, respectively.

The remaining parameters of Proposed 1 are the same as in CS-VBM3D. As a result, at each stage of VBM3D 24 different transforms can be randomly selected, i.e., \( E = 24^2 = 576 \) different combinations are possible. At each iteration we define 6 random indices in order to define the transforms for the both stages. For example, in order to define transform type \( \Psi_{HD} \), at each iteration we randomly select the transform type index using MATLAB as \( k_{index} = \text{rand}([1, 3]) \). Then, \( \Psi_{WD} \) is selected as 'no transform', 'dct2' or 'bior1.5', if \( k_{index} \) is 1, 2 or 3, respectively.

5 Experimental Results

Experimental results were obtained for the test video sequences * "Akiyo", "Carphone", "Coastguard", "Container", "Foreman", "Hall", "Mother-daughter", "Paris" and "Silent"* with frame resolution 352 x 288. In all cases we used 256 x 256 top left part of frames 20, 21, ..., 36.

Tables 3–6 show recovery performance for different approaches for noiselet-based sensing model introduced in Section 2.1, while Tables 7–10 show recovery performance for different approaches for block-based sensing model introduced in Section 2.2. The both sensing models were used with parameters \( G = 8 \) and \( \alpha = 3.5 \). For recovery performance comparison we used the following state-of-the-art algorithms:

- CS-Asif is Asif et al. [7] reconstruction based on convex optimization. We used source code available at the author web page 1. To maximize the recovery performance we utilized 2-D dual-tree wavelet transform as the sparsity basis, 5 motion adaptation iterations and applied the recovery for the whole video sequence to avoid any border effects.

*http://media.xiph.org/video/derf/
1http://users.ece.gatech.edu/sasif/
To solve minimization problem (22) we used NESTA toolbox \(^\dagger\) with the parameters \(\mu_f = 10^{-5}\), \(\delta = 0.01\) and NESTAAmaxiter = 25. Proposed 1 is the proposed algorithm, where the thresholding is performed by VBM3D with the random sparsifying transform selection introduced in Section 4.2.

Proposed 2 is an extension of Proposed 1, where at each iteration \(k\), the residual frames are calculated as \(\Delta x^k = c \Psi \alpha\), where \(\Psi\) is a spatially sparsifying transform (e.g., wavelet transform) and \(\alpha\) corresponds to the following TV minimization program:

\[
\begin{align*}
\min_{\alpha} & \quad \| \alpha \|_{TV} \\
\text{subject to} & \quad \| \Delta y^k - \Phi \Psi \alpha \|_2 < \delta.
\end{align*}
\]

To solve minimization problem (22) we used NESTA toolbox \(^\dagger\) with the parameters \(\mu_f = 10^{-5}\), \(\delta = 0.01\) and NESTAAmaxiter = 25.

Proposed 1 is the proposed algorithm, where the thresholding is performed by VBM3D with the random sparsifying transform selection introduced in Section 4.2.

Proposed 2 is an extension of Proposed 1, where at each iteration \(k\), the residual frames are calculated as \(\Delta x^k = \Phi^\dagger \Delta y^k\) with probability \(\theta = 2/3\) and as \(\Delta x^k = \alpha\) with probability \(1 - \theta\), where \(\alpha\) corresponds to the following TV minimization program:

\[
\begin{align*}
\min_{\alpha} & \quad \| \alpha \|_{TV} \\
\text{subject to} & \quad \| \Delta y^k - \Phi \Psi \alpha \|_2 < \delta.
\end{align*}
\]

For the following total variation (TV) minimization problem we used NESTA toolbox with the parameters \(\mu_f = 10^{-5}\), \(\delta = 0.01\) and NESTAAmaxiter = 25. For CS-VBM3D, Proposed 1 and Proposed 2 we used \(\beta = 1.75\) and \(\sigma_0\) and \(K\) given by Table 2.

All the recovery algorithms have been implemented using MATLAB and used identical measurement values corresponding to the same realization of a random generator.

The presented results show that utilization of the random sparsifying transforms in Proposed 1 outperforms CS-VBM3D in 56 out of 72 experiments. In average, it provides from 0.19 to 0.54 dB gain in PSNR, while in some cases from 1 to 2 dB gain in PSNR is achieved. Herewith, in average Proposed 1 provides from 0.91 to 6.19 dB gain in PSNR comparing to CS-Asif, from 0.77 to 4.5 dB gain in PSNR comparing to CS-Zhao and from 1.66 to 3.12 dB gain in PSNR comparing to CS-Kim. Proposed 2 exploits the randomly selected algorithm for the frame residual computation, i.e., it extends the idea utilized in Proposed 1. As a result, in average Proposed 2 outperforms Proposed 1 from 0.39 to 1.24 dB. Similar observations could be made using SSIM quality metric. As an example, Figures 14–15 show reconstructed second frame for ‘Container’ and “Foreman”, respectively, for for Block-based sensing model at rate 1/30.

Table 11 shows an average time needed for reconstruction of one frame in MATLAB by Processor Intel(R) Core(TM) i7 2.8GHz. We can see that Proposed 1 is 3.8, 9.6 and 5.7 times faster than CS-Asif, CS-Kim and CS-Zhao, respectively, and 28% slower than CS-VBM3D. Proposed 2 is 5.9, 4.6 and 1.2 times slower than CS-VBM3D, Proposed 1 and CS-Asif, respectively, and 1.24 and 3.4 times faster than CS-Zhao and CS-Kim, respectively. Herewith, its complexity could be reduced by increase of parameter \(\vartheta\).

6 Conclusion

A compressive sensed video recovery via iterative thresholding has been considered. Using an illustrative examples when 2-D DCT with different block size, 2-D DCT with different shift of the blocks grid and different 2-D DWTs are used as sparsifying transforms, we showed that the random selection of transform at each iteration notably improves the performance of the reconstruction in comparison with a case, when a single transform is used, without significant increase of computational complexity. Then we extended this approach for a case when VBM3D with random transform selection is used as the thresholding operator and demonstrated that up to 0.54 dB improvement in PSNR is achieved compared to VBM3D with a single transform defined in its "High Profile". Finally, we proposed to randomly select a frame residual computation algorithm as well and showed that the random selection between simple back projection and total variation minimization provides additional gain up to 1.24 dB in PSNR. We also demonstrated that the proposed recovery algorithm outperforms existing the state-of-the-art algorithms for both global and block-based sensing models. It makes the proposed approach attractive for image and video recovery from compressive measurements.

Acknowledgment

E. Belyaev acknowledges the support of the Government of the Russian Federation through the ITMO Fellowship and Professorship Program. M. Codreanu would like to acknowledge the support of the European Union’s Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Grant Agreement No. 793402 (COMPRESS NETS). M. Juntti acknowledges the support of Academy of Finland 6Genesis Flagship (grant no. 318927).
Table 3: PSNR and (SSIM) for different recovery algorithms for Noiselet sensing model at rate 1/5

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent</td>
<td>34.57 (0.9053)</td>
<td>34.44 (0.9384)</td>
</tr>
<tr>
<td>Paris</td>
<td>29.97 (0.8271)</td>
<td>34.14 (0.9508)</td>
</tr>
<tr>
<td>Container</td>
<td>30.33 (0.8039)</td>
<td>31.18 (0.8290)</td>
</tr>
<tr>
<td>Average</td>
<td>34.68 (0.8956)</td>
<td>36.30 (0.9228)</td>
</tr>
</tbody>
</table>

Table 4: PSNR and (SSIM) for different recovery algorithms for Noiselet sensing model at rate 1/10

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent</td>
<td>34.57 (0.9053)</td>
<td>34.44 (0.9384)</td>
</tr>
<tr>
<td>Paris</td>
<td>29.97 (0.8271)</td>
<td>34.14 (0.9508)</td>
</tr>
<tr>
<td>Container</td>
<td>30.33 (0.8039)</td>
<td>31.18 (0.8290)</td>
</tr>
<tr>
<td>Average</td>
<td>34.68 (0.8956)</td>
<td>36.30 (0.9228)</td>
</tr>
</tbody>
</table>

Table 5: PSNR and (SSIM) for different recovery algorithms for Noiselet sensing model at rate 1/20

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent</td>
<td>34.57 (0.9053)</td>
<td>34.44 (0.9384)</td>
</tr>
<tr>
<td>Paris</td>
<td>29.97 (0.8271)</td>
<td>34.14 (0.9508)</td>
</tr>
<tr>
<td>Container</td>
<td>30.33 (0.8039)</td>
<td>31.18 (0.8290)</td>
</tr>
<tr>
<td>Average</td>
<td>34.68 (0.8956)</td>
<td>36.30 (0.9228)</td>
</tr>
</tbody>
</table>

Table 6: PSNR and (SSIM) for different recovery algorithms for Block-based sensing model at rate 1/5

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent</td>
<td>34.57 (0.9053)</td>
<td>34.44 (0.9384)</td>
</tr>
<tr>
<td>Paris</td>
<td>29.97 (0.8271)</td>
<td>34.14 (0.9508)</td>
</tr>
<tr>
<td>Container</td>
<td>30.33 (0.8039)</td>
<td>31.18 (0.8290)</td>
</tr>
<tr>
<td>Average</td>
<td>34.68 (0.8956)</td>
<td>36.30 (0.9228)</td>
</tr>
</tbody>
</table>

Table 7: PSNR and (SSIM) for different recovery algorithms for Block-based sensing model at rate 1/10

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent</td>
<td>34.57 (0.9053)</td>
<td>34.44 (0.9384)</td>
</tr>
<tr>
<td>Paris</td>
<td>29.97 (0.8271)</td>
<td>34.14 (0.9508)</td>
</tr>
<tr>
<td>Container</td>
<td>30.33 (0.8039)</td>
<td>31.18 (0.8290)</td>
</tr>
<tr>
<td>Average</td>
<td>34.68 (0.8956)</td>
<td>36.30 (0.9228)</td>
</tr>
</tbody>
</table>
### Table 8: PSNR and (SSIM) for different recovery algorithms for Block-based sensing model at rate 1/10

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Akiyo</td>
<td>34.74 (0.9581)</td>
<td>34.75 (0.9544)</td>
<td>37.64 (0.9671)</td>
<td>37.71 (0.9679)</td>
<td>37.96 (0.9698)</td>
</tr>
<tr>
<td>Carphone</td>
<td>34.21 (0.9365)</td>
<td>32.14 (0.9071)</td>
<td>35.44 (0.9400)</td>
<td>35.94 (0.9429)</td>
<td>36.23 (0.9457)</td>
</tr>
<tr>
<td>Coastguard</td>
<td>24.19 (0.6584)</td>
<td>25.47 (0.6753)</td>
<td>27.52 (0.7502)</td>
<td>28.20 (0.7647)</td>
<td>28.74 (0.7911)</td>
</tr>
<tr>
<td>Container</td>
<td>22.88 (0.7877)</td>
<td>25.61 (0.8199)</td>
<td>27.70 (0.8617)</td>
<td>29.73 (0.8805)</td>
<td>30.20 (0.8883)</td>
</tr>
<tr>
<td>Foreman</td>
<td>31.19 (0.9122)</td>
<td>32.12 (0.8913)</td>
<td>33.78 (0.9166)</td>
<td>34.84 (0.9233)</td>
<td>35.56 (0.9308)</td>
</tr>
<tr>
<td>Hall</td>
<td>24.06 (0.8514)</td>
<td>26.80 (0.8827)</td>
<td>30.11 (0.9295)</td>
<td>30.96 (0.9288)</td>
<td>31.97 (0.9366)</td>
</tr>
<tr>
<td>Mother-daughter</td>
<td>39.11 (0.9566)</td>
<td>36.86 (0.9352)</td>
<td>38.95 (0.9555)</td>
<td>38.57 (0.9524)</td>
<td>39.19 (0.9575)</td>
</tr>
<tr>
<td>Paris</td>
<td>20.16 (0.7171)</td>
<td>22.40 (0.7474)</td>
<td>26.04 (0.8714)</td>
<td>26.69 (0.8682)</td>
<td>26.91 (0.8726)</td>
</tr>
<tr>
<td>Silent</td>
<td>30.99 (0.8436)</td>
<td>29.89 (0.8012)</td>
<td>31.28 (0.8458)</td>
<td>31.12 (0.8420)</td>
<td>31.44 (0.8512)</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>29.06 (0.8468)</td>
<td>29.56 (0.8451)</td>
<td>32.05 (0.8931)</td>
<td>32.64 (0.8967)</td>
<td>33.13 (0.9048)</td>
</tr>
</tbody>
</table>

### Table 9: PSNR and (SSIM) for different recovery algorithms for Block-based sensing model at rate 1/20

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Akiyo</td>
<td>26.68 (0.8722)</td>
<td>28.91 (0.8705)</td>
<td>31.94 (0.9275)</td>
<td>32.25 (0.9231)</td>
<td>33.30 (0.9374)</td>
</tr>
<tr>
<td>Carphone</td>
<td>24.59 (0.8224)</td>
<td>25.60 (0.7791)</td>
<td>29.01 (0.8754)</td>
<td>29.81 (0.8851)</td>
<td>31.29 (0.9041)</td>
</tr>
<tr>
<td>Coastguard</td>
<td>20.20 (0.4713)</td>
<td>21.75 (0.4732)</td>
<td>23.16 (0.5436)</td>
<td>24.44 (0.6100)</td>
<td>25.14 (0.6527)</td>
</tr>
<tr>
<td>Container</td>
<td>18.90 (0.6405)</td>
<td>21.58 (0.6637)</td>
<td>23.60 (0.7501)</td>
<td>24.64 (0.7662)</td>
<td>25.37 (0.7958)</td>
</tr>
<tr>
<td>Foreman</td>
<td>28.21 (0.8667)</td>
<td>26.16 (0.7991)</td>
<td>29.92 (0.8768)</td>
<td>30.77 (0.8864)</td>
<td>31.92 (0.8986)</td>
</tr>
<tr>
<td>Hall</td>
<td>20.35 (0.7243)</td>
<td>21.90 (0.7296)</td>
<td>24.23 (0.8240)</td>
<td>24.64 (0.8153)</td>
<td>25.87 (0.8476)</td>
</tr>
<tr>
<td>Mother-daughter</td>
<td>30.37 (0.8972)</td>
<td>32.51 (0.8813)</td>
<td>35.12 (0.9215)</td>
<td>35.15 (0.9176)</td>
<td>36.14 (0.9295)</td>
</tr>
<tr>
<td>Paris</td>
<td>15.11 (0.4446)</td>
<td>18.26 (0.5244)</td>
<td>20.91 (0.6787)</td>
<td>21.53 (0.6843)</td>
<td>21.95 (0.7082)</td>
</tr>
<tr>
<td>Silent</td>
<td>26.23 (0.7210)</td>
<td>26.33 (0.6766)</td>
<td>28.49 (0.7595)</td>
<td>28.07 (0.7427)</td>
<td>28.95 (0.7750)</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>23.40 (0.7178)</td>
<td>24.78 (0.7108)</td>
<td>27.38 (0.7952)</td>
<td>27.90 (0.8034)</td>
<td>28.88 (0.8277)</td>
</tr>
</tbody>
</table>

### Table 10: PSNR and (SSIM) for different recovery algorithms for Block-based sensing model at rate 1/30

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Akiyo</td>
<td>25.48 (0.8330)</td>
<td>26.55 (0.8119)</td>
<td>29.50 (0.8867)</td>
<td>28.76 (0.8615)</td>
<td>30.24 (0.8971)</td>
</tr>
<tr>
<td>Carphone</td>
<td>23.69 (0.7922)</td>
<td>22.83 (0.6846)</td>
<td>25.53 (0.7959)</td>
<td>25.92 (0.7887)</td>
<td>27.69 (0.8392)</td>
</tr>
<tr>
<td>Coastguard</td>
<td>17.03 (0.3662)</td>
<td>20.49 (0.3954)</td>
<td>21.86 (0.4679)</td>
<td>22.60 (0.5148)</td>
<td>23.30 (0.5611)</td>
</tr>
<tr>
<td>Container</td>
<td>18.30 (0.5928)</td>
<td>20.21 (0.6075)</td>
<td>21.94 (0.6791)</td>
<td>22.68 (0.6953)</td>
<td>23.32 (0.7234)</td>
</tr>
<tr>
<td>Foreman</td>
<td>24.35 (0.8110)</td>
<td>22.79 (0.7285)</td>
<td>24.99 (0.8227)</td>
<td>25.94 (0.8290)</td>
<td>27.81 (0.8544)</td>
</tr>
<tr>
<td>Hall</td>
<td>18.61 (0.6344)</td>
<td>20.25 (0.6490)</td>
<td>22.45 (0.7532)</td>
<td>22.47 (0.7332)</td>
<td>23.47 (0.7748)</td>
</tr>
<tr>
<td>Mother-daughter</td>
<td>28.18 (0.8556)</td>
<td>29.19 (0.8296)</td>
<td>32.40 (0.8898)</td>
<td>32.30 (0.8826)</td>
<td>34.13 (0.9060)</td>
</tr>
<tr>
<td>Paris</td>
<td>15.69 (0.4218)</td>
<td>17.18 (0.4408)</td>
<td>19.33 (0.5724)</td>
<td>19.72 (0.5780)</td>
<td>20.36 (0.6168)</td>
</tr>
<tr>
<td>Silent</td>
<td>24.42 (0.6575)</td>
<td>24.56 (0.6031)</td>
<td>27.01 (0.7052)</td>
<td>26.28 (0.6705)</td>
<td>27.52 (0.7237)</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>21.75 (0.6627)</td>
<td>22.67 (0.6389)</td>
<td>25.00 (0.7303)</td>
<td>25.19 (0.7282)</td>
<td>26.43 (0.7663)</td>
</tr>
</tbody>
</table>
Fig. 14: Performance comparison for Block-based sensing model at rate $1/30$ for "Container", a) Original frame b) CS-Zhao [5], c) CS-Kim [20], d) CS-VBM3D [21], e) Proposed 1, f) Proposed 2

Fig. 15: Performance comparison for Block-based sensing model at rate $1/30$ for "Foreman", a) Original frame b) CS-Zhao [5], c) CS-Kim [20], d) CS-VBM3D [21], e) Proposed 1, f) Proposed 2
Table 11 Execution time comparisons

<table>
<thead>
<tr>
<th>Recovery algorithm</th>
<th>Average recovery time per frame, sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS-Kim [20]</td>
<td>1736</td>
</tr>
<tr>
<td>CS-Zhuo [5]</td>
<td>1391</td>
</tr>
<tr>
<td>CS-VRM3D [21]</td>
<td>190</td>
</tr>
<tr>
<td>Proposed 1</td>
<td>243</td>
</tr>
<tr>
<td>Proposed 2</td>
<td>1119</td>
</tr>
</tbody>
</table>

7 References


Appendix 1. The Sensing Implementation

As it was mentioned in Section 2.1, operations such as \( y = \Phi x \) or \( x = \Phi^T y \) can be performed without matrix multiplication. First, for an image \( X \) size of \( N \times N \) the downsampling via one-level Haar transform is \( Y = X W X W^T \), where \( W = \frac{1}{\sqrt{2}} [I_{N/2} \otimes [1]] \), \( I_{N/2} \) is \( N/2 \times N/2 \) identity matrix. Such matrix multiplication can be implemented via addition of adjacent elements, i.e., \( X W X W^T \) can be performed via \( 3/4 N^2 \) additions only. Second, the dragon noiselet transform is performed via Algorithm 4 [6, 7]. One can see that it also does not need any multiplications and requires \( 2 N^2 \log_2 N \) additions or subtractions. Additionally, after the forward or inverse noiselet transform, all elements should be divided by \( N \). But this operation is needed only for small portion of coefficients selected as measurements. As a result, this sensing model is low complex and guarantees fast preview of a low-resolution image.

Algorithm 4: Dragon noiselet transform (forward and inverse)

Input: \( x \)

1. \( c \leftarrow N^2 - 1 \)
2. for \( j = 0, \ldots, N^2/2 \)
3. \( k \leftarrow j + c \)
4. \( y_k \leftarrow y_j + y_k \)
5. \( y_k \leftarrow y_j - y_k \)
6. end for
7. \( d \leftarrow N^2/2 \)
8. for \( j = 0, \ldots, N^2 \)
9. \( k \leftarrow j + c + d \)
10. \( t \leftarrow y_j \)
11. \( y_j \leftarrow y_j - y_k \)
12. \( y_k \leftarrow t + y_k \)
13. end for
14. end for

Appendix 2. The Euclidean Norm of the Equivalent Sensing Matrix

By the definition, the Euclidean norm of the equivalent sensing matrix \( \tilde{\Phi} \) is

\[
\| \tilde{\Phi} \|_2 = \sqrt{\lambda_{\max}(\tilde{\Phi}^T \tilde{\Phi})},
\]

(22)

where \( \lambda_{\max}(\cdot) \) denotes the maximum eigenvalue of a matrix. Utilizing (10), the matrix \( \tilde{\Phi}^T \tilde{\Phi} \) is

\[
\tilde{\Phi}^T \tilde{\Phi} = \begin{bmatrix} W & N_\lambda - N_\lambda W^T W \\ N_\lambda - N_\lambda W^T W & 0 \end{bmatrix} \begin{bmatrix} W^T N_\lambda^T - W^T W^T N_\lambda \\ 0 \end{bmatrix}
\]

(23)

i.e., it is a block diagonal matrix. Therefore, its eigenvalues are given by the eigenvalues of each block. Let us consider the maximum eigenvalue for each block. For the first block the maximum eigenvalue is \( \lambda_{\max}(I) = 1 \). Since \( N_\lambda W^T W^T \) is symmetric positive semidefinite matrix, the eigenvalues of the second block \( I - N_\lambda W^T W^T \) are between \( 1 - \lambda_{\max}(N_\lambda W^T W^T) \) and 1. Taking into account that \( N_\lambda \) and \( W \) are submatrices of an orthogonal matrices, i.e., \( N_\lambda N_\lambda^T = I \) and \( WW^T = I \), we obtain

\[
\lambda_{\max}(N_\lambda W^T W^T N_\lambda) \leq \| N_\lambda W^T W^T N_\lambda \|_2 \leq \| N_\lambda \|_2^2 \| W^T W^T \|_2 = 1.
\]

Therefore, the maximum eigenvalue of \( I - N_\lambda W^T W^T N_\lambda \) is not higher than 1. Taking into account the maximum eigenvalues of the considered blocks, \( \lambda_{\max}(\tilde{\Phi}^T \tilde{\Phi}) = 1 \), i.e., \( \| \tilde{\Phi} \|_2 = 1 \).