J Real-Time Image Proc (2019) 16:1629–1642
DOI 10.1007/s11554-016-0659-y

ORIGINAL RESEARCH PAPER

Real-time video denoising on multicores and GPUs with Kalman-based and Bilateral filters fusion

Sergio G. Pfleger[1] · **Patricia D. M. Plentz**[1] · **Rodrigo C. O. Rocha**[2] · **Alyson D. Pereira**[1] · **Márcio Castro**[1]

Received: 17 May 2016 / Accepted: 28 November 2016 / Published online: 8 February 2017

Abstract In the context of video processing, image noise caused by acquisition, transfer and image compression can be attenuated by video denoising algorithms. However, their computational cost must be as low as possible to allow them to be applied to real-time applications. In this paper, we propose STMKF, a real-time video denoising algorithm based on Kalman and Bilateral filters. We evaluate the effectiveness of STMKF using several common videos used in the literature and we compare it to other denoising algorithms using both the PSNR and SSIM metrics. Our experimental results show that STMKF is competitive with other filters, especially for videos that feature stationary backgrounds such as in videoconferencing, video lectures and video surveillance. We also evaluate the performance of our parallel implementations of STMKF for CPUs and GPUs. STMKF achieved a performance improvement of up to 2.9× on a Intel i7 multicore processor with 4 cores compared to the sequential solution. The results obtained with the GPU version of STMKF on a NVIDIA Tesla K40 showed a performance improvement of up to 7.6× compared to the Intel i7 multicore processor.

Keywords Spatiotemporal video denoising · Kalman filter · Bilateral filter · Multicore · GPU

Electronic supplementary material The online version of this article (doi:10.1007/s11554-016-0659-y) contains supplementary material, which is available to authorized users.

✉ Patricia D. M. Plentz
patricia.plentz@ufsc.br

Sergio G. Pfleger
sergiogenilson@gmail.com

Rodrigo C. O. Rocha
rcor@pucminas.br

Alyson D. Pereira
alyson.pereira@posgrad.ufsc.br

Márcio Castro
marcio.castro@ufsc.br

[1] Department of Informatics and Statistics (INE), Federal University of Santa Catarina (UFSC), Campus Universitário Reitor João David Ferreira Lima, Trindade, Cx.P. 476, Florianópolis 88040-900, Brazil

[2] Computer Science Department, Pontifical Catholic University of Minas Gerais (PUC Minas), Avenida Dom José Gaspar, 500, Belo Horizonte 30535-610, Brazil

1 Introduction

In the context of video processing, image noise can hardly be avoided. Because of that, several filters have been developed to reduce the noise caused by acquisition, transfer and image compression processes. They differ in several aspects such as the image quality obtained when applying the filter and their computational complexity. Typically, best visual results are achieved at the cost of heavy computational cost.

The computational cost of video denoising algorithms must be as low as possible to be suitable for video capture and transmission in real-time applications (e.g., videoconferencing, video lectures and video surveillance of private areas). In this paper, we present a video denoising algorithm suitable for real-time applications called STMKF. Our algorithm combines a time-domain filter (Kalman) with a space-domain filter (Bilateral) to achieve decent visual results with low computational costs. We discuss the main ideas behind STMKF and show how it was parallelized to exploit the parallelism available in modern processors such as general purpose and embedded multicores as well as graphics processing units (GPUs).

Overall, this paper presents the following main contributions:

- We propose a fast real-time video denoising algorithm called STMKF, which combines Kalman and Bilateral filters;
- We propose modifications to the original Kalman Filter that allow it to converge faster on moving regions;
- We simplify the use of the Kalman Filter by automatizing one of its main input parameters;
- We propose parallel solutions of STMKF for multicores and GPUs. Our GPU implementation exploits asynchronous data transfers to overlap communications with computations.

The remainder of this work is organized as follows. Section 2 presents an overview of the Kalman and Bilateral filters used by our approach. Section 3 discusses related works. The STMKF algorithm and its parallel versions are presented in Sect. 4. The effectiveness and performance of STMKF are discussed in Sects. 5 and 6, respectively. Finally, Sect. 7 concludes this paper.

2 Background

2.1 Image noise and filters

Image noise is a variation (random or not) in brightness or color of an image, which is not present in the imaged object [11]. Errors in the image acquisition, compression and/or transmission processes are usually the main sources of noise. Although undesirable in computer vision processes, image noise can hardly be avoided. One of the most common types of noise in image capture is Gaussian noise (GN) [1, 12]. GN is caused by random fluctuations in the signal reading. The probability density is similar to a normal distribution [1] and it is independent for each pixel and from the signal intensity [11].

In the signal processing domain, filters are used to remove unwanted components or signal characteristics. In this paper, and however, a filter is considered as a process where the image noise is removed, resulting in an image with values near to the real ones of the imaged object. Many denoising techniques have been developed in the literature [5, 7, 8, 37]. Several of these techniques are based on Bilateral and Kalman filters, which are well-known denoising solutions. In the next section, we will further describe them.

2.2 Bilateral filter

The Bilateral filter (BF) is a spatial nonlinear, edge- preserving and noise-reducing smoothing filter for images proposed in [37]. In BF, each pixel of the image is replaced by a weighted average on their neighborhood pixels, considering the geometric closeness and their photometric similarity. Chaudhury [5] proposes a $O(n)$ implementation of the BF, where n represents the number of pixels of the input image.

BF has three main parameters that have a significant impact on the final result: the diameter of each pixel neighborhood (d), the photometric propagation factor (σ_c) and the geometric propagation factor (σ_s). A larger value of σ_c means that farther colors within the pixel neighborhood will be mixed together, resulting in larger areas of semi-equal color. A larger value of σ_c, on the other hand, means that farther pixels will influence each other as long as their colors are close enough. Figure 1 shows how these parameters affect the final result.

Although BF is computationally efficient, it presents unsatisfactory results when used for video denoising. Since it is a spatial denoising method, the image noise reduction is applied to each frame individually without taking into account previous frames. This may result in local flickering artifacts [10], which is a change of brightness on small areas that may be unpleasant to the eyes.

2.3 Kalman filter

The Kalman filter (KF) was originally developed by Rudolf Kalman in 1960 [17] and uses measurements contaminated by uncertainties. It is a temporal filter that intends to approximate the acquired data (probably containing noise) to the real data (without noise). In this paper, we adopt a simple case with a trivial prediction step of the most general expression of KF.

Basically, the filter performs two main steps, prediction and correction, for each new data acquisition. In the first step, it estimates the next state x_k^- (1) and its covariance P_k^- (2) based on its previous state (x_{k-1}) and previous covariance (P_{k-1}), respectively. In the second step, it computes the Kalman gain K_k (3). Then, it updates the estimated state x_k (4) using K_k, x_k^- and the acquired data z_k. Finally, it computes the estimated covariance P_k that will be used in the next state (5).

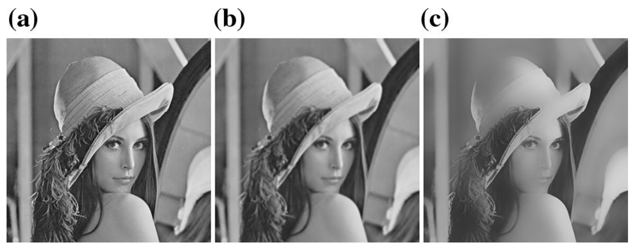

Fig. 1 Example of the impacts of BF parameters. **a** Original image. **b** $d = 6$, $\sigma_c = 100$ and $\sigma_s = 100$. (c) $d = 50$, $\sigma_c = 50$ and $\sigma_s = 50$

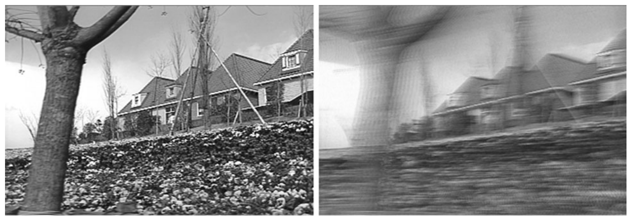

Fig. 2 Original frame of the *garden* sequence (*left*) and the resultant frame after applying KF (*right*)

$$x_k^- = x_{k-1} \tag{1}$$

$$P_k^- = P_{k-1} + Q \tag{2}$$

$$K_k = P_k^-(P_k^- + R)^{-1} \tag{3}$$

$$x_k = x_k^- + K_k(z_k - x_k^-) \tag{4}$$

$$P_k = (1 - K_k)P_k^- \tag{5}$$

It is worth noting that both Q and R are constants and the choice of their values modifies the filter strength. If Q is fixed, fluctuations of z_k will affect x_k less intensively (when R is increased) or more intensively (when R is decreased). Likewise, if R is fixed, fluctuations of z_k affect x_k more intensively (when Q is increased) or less intensively (when Q is decreased).

The drawback of KF is to present resistance to abrupt changes in data acquisition. Consider that a sensor is reading a real data r_1 until the instant t. After instant t, the real data suddenly change to a new value r_2. Because of noises from the video capturing process, until the instant t, the values being read by the sensor will oscillate around r_1, and after the instant t the values being read will oscillate around r_2. Until the instant t, the KF will approximate the read values to the real value r_1, but shortly afterward, instead of approximating the read values to the new real data r_2, the KF will still approximate to the previous value r_1, taking a few cycles of video capturing until the processed value converges to the new real value r_2.

The problem described above occurs in videos when KF is applied. Whenever an object moves in an image sequence, a step occurs in the pixel values. As KF presents resistance to converge to a new pixel value when steps occur in the capture of the signal, some ghosts in the image are formed, as shown in Fig. 2.

3 Related work

In a general way, video denoising methods can be divided in three groups: methods based on a transform [24, 30, 33, 36, 42], methods that use block matching and motion detection jointly [7, 8, 22, 23] and methods that combine spatial and temporal domains [4, 6, 14, 37, 41, 43]. Obviously, there are video denoising methods that merge two or more techniques used in these three different groups but most video denoising algorithms belong to one of these three groups.

In the first group, Pizurica et al. [30] developed a sequential wavelet domain and temporal filtering scheme combined with optimized parameters (SEQWT). In the sequential spatiotemporal denoising proposed filter, motion detection and temporal filtering are performed over spatially denoised frames. The realization of a motion adaptive temporal filter benefits from the use of a high-quality spatial denoising. Rahman et al. [33] proposed a joint probability density function of the video wavelet coefficients for any two neighboring frames by using the bivariate Gaussian distribution. They show that the correlation coefficient of the density function gives an indirect measure of the motion that exists between any two frames. The proposed density function is then employed for spatial filtering of the noisy video wavelet coefficients. The spatially filtered coefficients are then passed through a recursive time averaging filter for additional noise reduction (we refer to this approach as IFSM). Selesnick and Li [36] presented the design and application of the non-separable oriented 3D dual-tree wavelet transform for video denoising (3DWTF). Motion-based multiscale decomposition for video is reached with this transform which isolates in its subbands motion along different directions. In addition, it is investigated the denoising of video using the 2D and 3D dual-tree oriented wavelet transforms, where the 2D transform is applied to each frame individually. Zlokolica et al. [42] proposed a video denoising method based on non-decimated wavelet band filtering (WRSTF). In the proposed method, motion estimation and adaptive recursive temporal filtering are performed in a closed loop, followed by an intra-frame spatially adaptive filter. The algorithm is implemented in three steps: motion estimation, motion compensation and the adaptive spatial filtering scheme. All computations occurs in the wavelet domain.

Considering the group of methods that use block matching and motion detection jointly, the Block-matching 3D Denoising (BM3D), proposed in [7], reduces noise by processing similar image blocks. Block-matching algorithm is used to find similar blocks based on a reference one. It stores the blocks in a 3D structure and constructs blocks using a weighted average. VBM3D algorithm [8] extends the ideas of BM3D for image sequences (video), finding identical blocks not only in space, but also in time. In [22, 23], the authors reduce noise in images sequences similarly as BM3D. They work with 3D blocks (3D patch), rather than BM3D 2D blocks, and put them in 4D structures. These filters, among others, are the state of the art of image and video denoising. However, the high computational complexity of these algorithms becomes prohibitive for real-time applications.

On the other hand, the group of methods that combine spatial and temporal domains present algorithms with lower computational complexity than other two groups. This feature is fundamental for real-time applications, such as videoconferencing systems, robotic navigation systems and surveillance systems. Zlokolica et al. [41] proposed a nonlinear filter which sorts pixels within a 3D window considering their difference with the central pixels value (3D KNN). The next step of this filter is to average the pixels in the window and then weighting them according to their sorting order. The proposed filter is an extension of that proposed in [27], which was first described by Davis and Rosenfeld [9]. The main advantage of the proposed filter is that it works well on video independently of the noise type (Gaussian and impulse, or a combination of both). Spatiotemporal varying filter (STVF) is proposed in [4] which could generate the best candidate value to replace the noisy value of current pixel by exploiting the corrections of its neighboring pixel values within a small region and taking optimal weights of them. STVF is able to produce optimal results in the sense that it minimizes the weighted least squared error. At the same time, STVF retains the sharpness of edges in object boundaries and it combines the advantages of conventional denoising filters that enable it to decrease the noise variance in smooth areas. Zuo et al. [43] performed an appropriate average filtering on current noisy frame to reduce the influence of noise. This step is useless to the final denoising result, but preparative to the motion estimation. Block-matching-based motion estimation is performed by comparing current pre-filtered frame with previously denoised frames. Then, the Kalman filter is applied on the current noisy frame, based on previous steps motion estimation results. On the other hand, the current noisy frame is also processed in the spatial domain by using the Bilateral filter, which aims at reducing the noise globally. Weighting of the two denoised frames showed a satisfactory result.

The Non-Local Means (NLM) algorithm [2, 3] exploits the fact that similar neighborhoods can occur several times anywhere in the image and can contribute for denoising. For a given target pixel, NLM computes the mean of all pixels in the image, weighted by how similar these pixels are to the target pixel. Han and Chen [13] combine NLM with the Kalman filter framework for video denoising. Although standard NLM algorithms are known to be computationally expensive [38], some variations reduce the computational complexity by reducing weights and neighborhood computations [18, 25]. Similarly, Jojy et al. [15] combine a variation of NLM called Discontinuity Adaptive Non-Local Means (DA-NLMF) with a variation of the Kalman filter (Importance Sampling Unscented Kalman filter).

Our filter (STMKF) fuses Blur, Bilateral and Kalman filters to generate the output denoised frame which is stored as the estimated value for the next frame. It combines spatial and temporal domains with lower computational complexity than other two groups. STMKF does not use motion estimation techniques (as it is proposed in the algorithms of the first group) because of the high computational complexity of these kind of techniques. The computational complexity of each algorithm step is $O(n)$, where n is the number of pixels, making it suitable for real-time applications.

4 Spatiotemporal fusion of Kalman-based and Bilateral filters (STMKF)

In Sect. 2.3, we presented an overview of the Kalman filter. In its original form, P, Q and R are covariance matrices, where the diagonal contains variances. In this paper, however, covariance matrices are simply variances, since we deal with scalar states. Because of that, we consider that all variables in STMKF represent 2D arrays containing $frame_{Height} \times frame_{Width}$ pixels and all numeric computations between them are performed by array operations that execute element by element operations (i.e., pixel-by-pixel operations).

As we mentioned in Sect. 2.3, KF relies on P_k to obtain K_k and to estimate the next state. However, when the scenario changes considerably, which is also when the ghost effect occurs, P_k^- must change accordingly to achieve better results. In (2), P_k^- consists of two components: P_{k-1} and Q. Considering a projection, it is necessary to maintain the component from the previous instant P_{k-1}. To allow KF to converge faster, we propose a modification to (2) as follows

$$P_k^- = P_{k-1} + \Delta^2 Q \tag{6}$$

where Δ is the difference between the average of a pixel's neighborhood at instants k and $k-1$. The average of a pixel's neighborhood is known as the *blur filter*. In this modified equation, when the scenario does not change significantly, Δ will be small. When motion occurs, on the other hand, the average of the neighborhood also changes and Δ will assume a higher value. Based on this observation, our STMKF algorithm will try to keep KF properties in motionless regions, whereas it will assume values closer to z_k for pixels with motion.

Indeed, (6) allows KF to converge faster. However, KF still relies on Q and R parameters. After performing several experiments varying R and Q parameters we observed the following: (1) when the video features low values of K, the best results are achieved when the value of R is high; and (2) analogously, when the video features higher values of K, the best results are achieved when the value of R is low. Based on that observation, we propose to automatize the choice of R as follows

$$R_k = 1 + R_{k-1}(1 + K_{k-1})^{-1} \tag{7}$$

In other words, R_k can be adjusted to regions with and without motion based on K_{k-1} and R_{k-1}. In (7), high values of K_{k-1} (i.e., values closer to 1) make the series converge for a value near 2. When K_{k-1} approaches 0, on the other hand, the series diverges, tending to $+\infty$. However, due to inherent characteristics of noise, Δ will not be equal to 0, which in turn implies that P^- will not be equal to 0, by equation (6). Thus, by (3) we have that K never equals 0. This way, R tends to stabilize at a value which usually allows for decent results.

Finally, spatial (BF) and temporal (KF) techniques are fused to achieve better results. Our solution for that relies on applying a weighted combination of x_k and the result of BF ($x_{bilateral}$) in the resultant frame $\hat{x_k}$, as expressed by

$$\hat{x}_k = (1 - K_k)x_k + K_k x_{bilateral} \tag{8}$$

The main idea behind this equation relies on the use of K_k to detect regions with motion (the values of K_k will be high in regions with motion), giving more weight to the Bilateral filter on these regions. This reduces the error introduced by the noise, so the affected pixels will have values closer to the real ones. Moreover, this second modification allows the Kalman filter to converge faster on moving regions, since $\hat{x}_k$ will also be used in the next frame as the estimation of the next state (x_k^-). Thus, (1) must be rewritten as $x_k^- = \hat{x}_{k-1}$ in order to take it into account.

Figure 3 shows one frame of the Salesman video sequence in different situations. We added to the original frame (Fig. 3a), a white additive Gaussian noise (Fig. 3b) and then applied the Bilateral (Fig. 3c) and Kalman (Fig. 3d) filters individually. As it can be observed, the Kalman filter presents good results on motionless regions, whereas it presents some ghosts where motion occurs. STMKF overcomes this issue (Fig. 3e), since it gives more weight to the Bilateral filter where motion occurs. Finally, Fig. 3f shows the values of K_k for this frame (the higher the value of K_k the brighter the color). As it can be observed, moving regions are much brighter than stationary ones.

In the next sections, we discuss the overview of STMKF as well as its parallel implementations for multicores and GPUs. We made our implementations publicly available[1] under the GPL 3 License, thus enabling other researchers to further enhance it.

[1] Source codes available at: http://github.com/sergiogenilson/STMKF.

4.1 Sequential algorithm

Figure 4 shows a graphical overview of the main operations performed by STMKF on each frame. It performs two operations taking the noised frame as input: (1) It computes the neighborhood average using the Blur filter; and (2) it computes BF. Next, Δ is obtained from the difference between the current blurred frame and the previous blurred frame. Then, it uses Δ and the noisy frame to calculate the denoised frame and K_k. After that, it computes the output denoised frame taking as input x_k, K_k and $x_{bilateral}$. Finally, the output denoised frame is stored as the estimated $\hat{x}_k$ for the next frame.

Algorithm 1 presents the pseudocode of the STMKF. We consider that the variable *video* stores all frames of a video sequence that shall be denoised. The main procedure (STMKF-MAIN) initializes some variables used by STMKF (recall that all variables are 2D arrays and all numeric operations on them are element by element operations). Then, it calls the core algorithm (STMKF-CORE) for each frame. Q is initialized with the values provided by the user (line 2), whereas the other 2D arrays are initialized with 0 ($\hat{x}$ and *pBlurred*), 0.5 (K) or 1 (P and R).

Algorithm 1 Sequential Algorithm

```
Global: $P, Q, R, K, \hat{x}, pBlurred$
 1: procedure STMKF-MAIN($video, q$)
 2:     $Q \leftarrow q$
 3:     $\hat{x}, pBlurred \leftarrow 0$
 4:     $K \leftarrow 0.5$
 5:     $P, R \leftarrow 1$
 6:     for each $frame$ of $video$ do
 7:         $blurred \leftarrow$ BLUR-FILTER($frame$)
 8:         $bf \leftarrow$ BILATERAL-FILTER($frame$)
 9:         $frameOUT \leftarrow$ STMKF-CORE($frame, blurred, bf$)
10:     end for
11: end procedure
12: function STMKF-CORE($frame, blurred, bf$)
13:     $\Delta \leftarrow pBlurred - blurred$
14:     $pBlurred \leftarrow blurred$
15:     $R \leftarrow 1 + R \cdot (1+K)^{-1}$
16:     $x^- \leftarrow \hat{x}$
17:     $P^- \leftarrow P + \Delta^2 \cdot Q$
18:     $K \leftarrow P^- \cdot (P^- + R)^{-1}$
19:     $x \leftarrow x^- + K \cdot (frame - x^-)$
20:     $\hat{x} \leftarrow (1-K) \cdot x + K \cdot bf$
21:     $P \leftarrow (1-K) \cdot P^-$
22:     return $\hat{x}$
23: end function
```

After the initialization, STMKF-MAIN gets the next frame (*frame*) from the input video sequence to be denoised (line 6). The frame is then used as input to the Blur and Bilateral filters (lines 7 and 8). The results of the Blur and Bilateral filters are stored in 2D arrays called *blurred* and *bf*, respectively. Finally, the original frame (*frame*) as well as

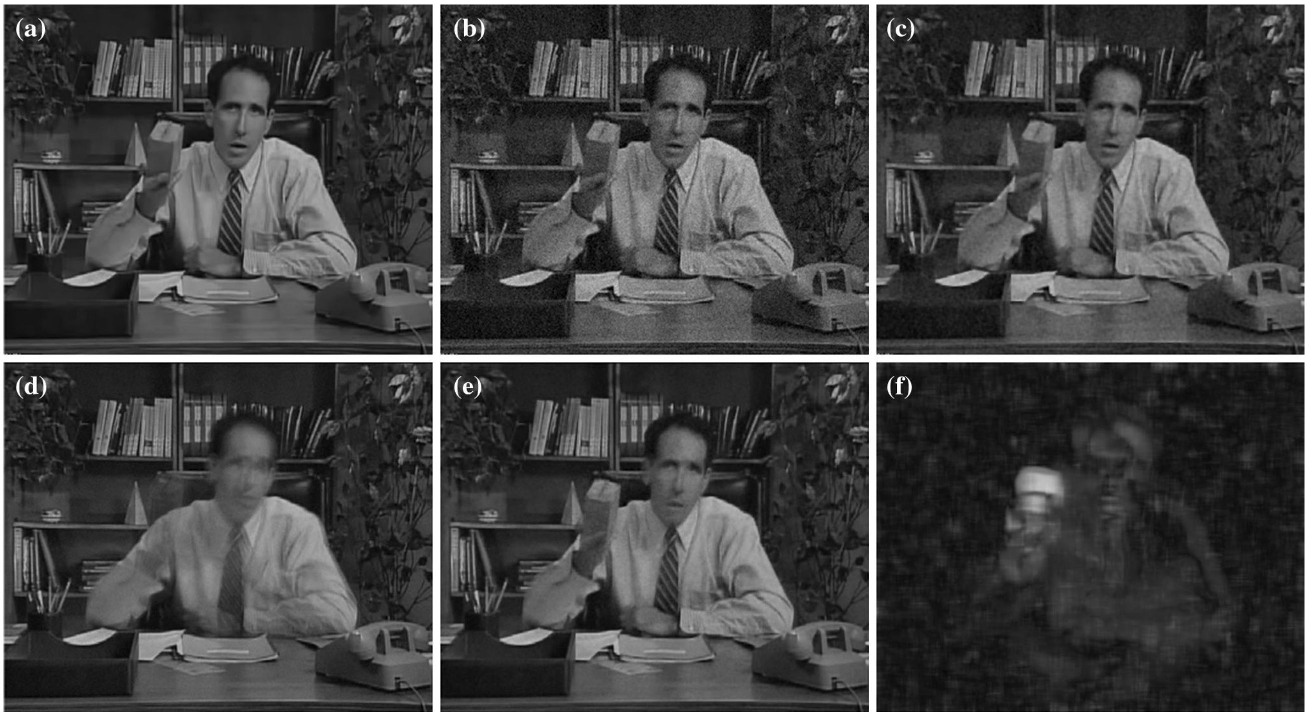

Fig. 3 Example of the Salesman video sequence: **a** original frame; **b** noised frame ($\sigma = 10$); **c** bilateral filter; **d** Kalman filter; **e** STMKF; and **f** values of K_k (the higher the value of K_k the *brighter* the color)

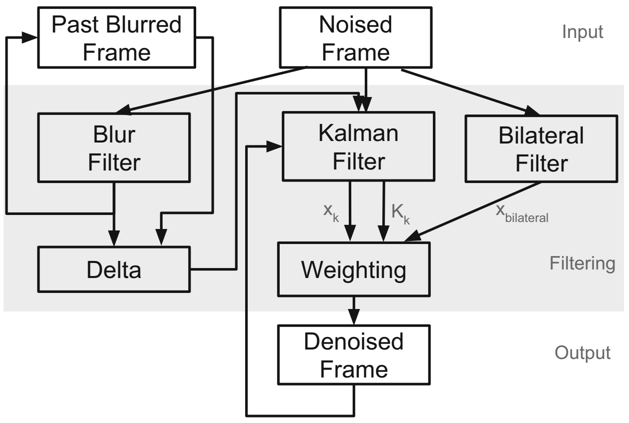

Fig. 4 STMKF diagram

pBlurred, *blurred* and *bf* is used as inputs to the main STMKF algorithm (STMKF-CORE), which will denoise the frame. The denoised frame is stored on *frameOUT*, which can be either stored on a file or shown on the screen. This process is repeated for each frame of the input video.

The STMKF fuses Blur, Bilateral and Kalman filters. There are efficient blurring algorithms suitable for real-time applications [34], for example, in our experiments we use the Box Blur filter which can be implemented with computational cost that is independent of kernel size, meaning that it is linear in the size of the image, no matter how much blurring is required [34]. Therefore, we can use a blurring algorithm of order $O(n)$, where n is the size of the image in pixels. There are also linear implementations, $O(n)$, for the BF algorithm [5]. As described in this section, the STMKF-CORE method applies only pixel-wise operations over 2D arrays. Therefore, the STMKF algorithm can be implemented with linear computational complexity, where n is the size of the image in pixels, for each frame of the input video.

We implemented the STMKF algorithm in C++. We rely on the Open Source Computer Vision Library (OpenCV) [32] to perform all array operations described in Algorithm 1. OpenCV is a library that features highly optimized abstractions and functions that ease the development of real-time computer vision applications. In addition to the array operations, OpenCV also includes efficient implementations of Blur and Bilateral filters. We used them to compute *blurred* and *bf* in our STMKF implementation.

4.2 Multicore implementation (MT-STMKF)

We implemented a parallel version of the STMKF algorithm (MT-STMKF) to make full use of current multicore processors. We adopted a simple yet efficient strategy to parallelize STMKF. Figure 5 shows an example of the proposed parallelization strategy with 4 threads. Each frame is virtually divided into t blocks, where t represents the number of threads to be used (4 blocks/threads in this example).

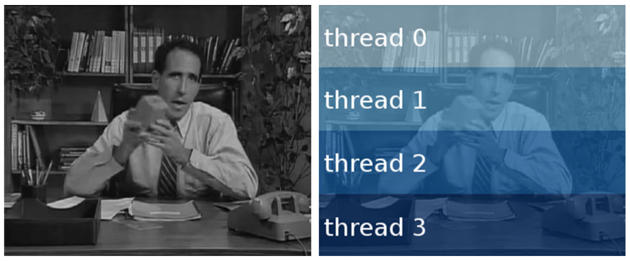

Fig. 5 Illustration of the parallel approach adopted in MT-STMKF. This example considers 4 threads, each one responsible for denoising 1 / 4 of each frame

Then, all threads execute the STMKF-CORE function on their own frame block in parallel.

We used a specific Application Programming Interface (API) called Open Multi-Processing (OpenMP) [28] to parallelize STMKF. OpenMP allows developers to add specific compiler directives to the original source code to determine portions of code that must be executed in parallel (*parallel regions*). Overall, the OpenMP API follows a *fork-join* model. The execution starts with a main thread (sequential execution). At runtime, whenever the main thread reaches a parallel region, the OpenMP runtime creates a set of worker threads to execute the code inside it in parallel. Then, all worker threads synchronize at end of a parallel region and the runtime resumes the sequential execution again.

OpenMP allowed us to easily implement the parallel strategy described above. Basically, we added a compiler directive to determine STMKF-CORE to be called inside a parallel region. Then, we assign a different frame block to each worker thread, which in turn calls STMKF-CORE using its own frame block as input. We could adopt the same parallelization strategy for the Blur and Bilateral filters. However, this was not necessary because OpenCV already includes parallel versions of them.

4.3 GPU implementation (GPU-STMKF)

GPUs are manycore processors optimized for highly parallel computation. Their design focuses to maximize the chip area and power budget dedicated to data processing rather than data caching and flow control, optimizing for the execution throughput of massive numbers of threads. The throughput-oriented design strives to maximize the total execution throughput of a large number of threads while allowing individual threads to take a potentially much longer time to execute [19].

GPUs require a particular parallel computing language such as Compute Unified Device Architecture (CUDA), which is specific for NVIDIA GPUs, and Open Computing Language (OpenCL), which works on a wide range of processors and GPUs. However, writing efficient code with CUDA or OpenCL is not trivial, since developers must pay special attention to a number of points, such as synchronization, data transfers and global / shared memory accesses. Fortunately, OpenCV features a GPU module to take advantage of such processors. The GPU module provides the user an explicit control on how data are moved between CPU and GPU memory. Although the user has to write some additional code to start using the GPU, this approach is both flexible and allows more efficient computations.

We used the features available in OpenCV to implement a GPU version of STMKF (GPU-STMKF). Basically, we had developed a GPU version of the STMKF-CORE function, so each frame could be denoised in the GPU in parallel. Again, we did not have to implement GPU versions of Blur and Bilateral filters, since the OpenCV framework already included them. Overall, there are three main steps to denoise each frame in GPU-STMKF: 1) upload the frame data from host memory to the GPU memory; 2) perform the blur, bilateral and STMKF-CORE computations in the GPU; and 3) download the processed frame back to the host memory.

In our first version of GPU-STMKF, data transfers between host/GPU and computations performed in the GPU were completely synchronous. This means that the frame $f + 1$ is only uploaded to the GPU after downloading back the frame f from the GPU. To hide the time spent with data transfers, we implemented a second version of GPU-STMKF that performs asynchronous data transfers between the host and the GPU with CUDA streams. In this new version, we upload the frame $f + 1$ to the GPU while the GPU is computing the frame f. Analogously, we download the frame $f - 1$ processed by the GPU to the host while the GPU is computing the frame f.

4.4 Discussion

As described before, our approach combines BF with a modified version of KF. However, filters other than BF could also be combined with KF. As shown in Sect. 3, combinations of Non-Local Means (NLM) and KF have already been proposed [13, 15].

We also performed an experimental analysis using NLM instead of BF in STMKF to assess whether it would achieve better results. We evaluated the use of the OpenCV implementation of NLM with different values for the *templateWindowSize* and *searchWindowSize* parameters: (1) the default values suggested by OpenCV; (2) the best possible values for each parameter and video sequence to improve the quality of the results; and (3) the minimum possible values for each parameter to reduce its computational cost. This approach showed similar results in terms of quality and a considerable performance degradation in terms of FPS from 3× (with the minimum possible values for the NLM parameters) up to 8.5× (with the best possible values for the NLM parameters) when compared to our original version of STMKF. We believe that such performance degradation can be prohibitive for some real-time applications.

Finally, we also replaced BF in STMKF by other less complex filters such as the Blur [34], Gaussian Blur [34] and Median filters. However, those combinations showed worse results than the original version of STMKF.

5 Effectiveness of STMKF

The effectiveness of an image noise filter is usually evaluated using the peak signal-to-noise ratio (PSNR). The PSNR is calculated by

$$PSNR = 10 \times log \frac{m^2}{MSE} \tag{9}$$

where m is the maximum value of a pixel and MSE is the mean squared error [24]. The PSNR of a video sequence is the average of the PSNRs of each frame. A high PSNR indicates that the original video sequence (without noise) is very similar to the denoised one.

Another common result evaluation method is through structural similarity (SSIM) [39]. SSIM is obtained by

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{10}$$

where x and y are image patches extracted from the local window of the original image and the denoised image and μ, σ and σ_{xy} represent the mean, variance and cross-correlation computed for the local window, respectively. The variables $c_1 = (k_1G)^2$ and $c_2 = (k_2G)^2$ are applied to stabilize the division by a weak denominator, so that $k_1 = 0.01$, $k_2 = 0.03$ and G is the dynamic range of the pixel values (i.e., the maximum value that can be represented with a given number of pixels). In this case, $G = 2^{bits\ per\ pixel} - 1 = 2^8 - 1 = 255$.

We do the following to compute the SSIM of a video sequence. We first compute the SSIM of each frame by calculating the average of its local SSIMs. Then, the SSIM of a video sequence is the average of the SSIMs of each frame. A high SSIM (i.e., near 1) indicates that the original video is very similar to the denoised one.

We evaluate the effectiveness of STMKF using several common videos used in the literature. For each video, we added a white additive Gaussian noise and then applied STMKF. The Gaussian noise is the most common type of noise considered by the authors of related works [30, 33, 36, 41, 42]. However, STMKF is also capable of achieving good results on videos that present any kind of noise that satisfies the assumptions needed to compute Δ (i.e., the noise distribution should be similar to the normal distribution and the noise should be independent for each pixel) such as the shot noise. Moreover, we fixed the parameters for the Blur and Bilateral filters in all experiments as follows: we set the kernel size in the Blur filter to 5 and used $d = 3$, $\sigma_c = 50$ and $\sigma_s = 50$ for the Bilateral filter.

It is worth noting that the quality of the results obtained with STMKF rely on the right choice for the Q parameter. Overall, we obtained decent visual results on all videos considered in this paper with values of Q close to 0.02. For the PSNR and SSIM metrics, however, the best value for Q varied from 0.01 to 0.1 and it was mostly affected by the amount of movement in the videos. After a careful analysis we concluded that Q values close to 0.01 improve PSNR and SSIM metrics on videos that feature stationary backgrounds or smooth background movements (e.g., Salesman and MissA). On the other hand, Q values close to 0.1 improve both PSNR and SSIM metrics on videos that feature several moving regions (e.g., tennis, football and foreman).

Tables 1 and 2 compare the effectiveness of STMKF against other approaches that can be applied on real-time applications using the PSNR and SSIM metrics. Overall, STMKF is competitive with several filters found in the literature for both metrics, especially for videos that feature stationary backgrounds. The Salesman sequence is an example of such case, in which STMKF has SSIM equal to 0.955 with a standard deviation (noise) of 10, outperforming WRSTF [42], 3DWTF [37] and IFSM [34] algorithms.

For sequences with several moving areas, such as the garden sequence, STMKF was outperformed by the other algorithms. The main reason comes from the fact that they use motion estimation and/or compensation techniques. As we mentioned before, although these techniques increase the complexity of these algorithms they may improve the quality of video denoising filters. However, the overall quality relies on how good these techniques can estimate motion. The football sequence, for instance, also has several moving areas but STMKF achieved better results than 3D KNN [41], SEQWT [30] and IFSM [33] algorithms. In this specific case, the motion cannot be easily predicted, making it difficult to be estimated and/or compensated by these techniques.

We would like to emphasize that STMKF is able to preserve the edges of the original videos. After applying an edge detection algorithm (Canny) on all videos considered in this paper, we observed that the edges found by Canny in both original and denoised frames (STMKF) were very similar. Finally, it is important to mention that STMKF achieves

Table 1 PSNR results compared to other algorithms

Test sequence	Noise	3D KNN [41]	SEQWT [30]	IFSM [33]	STMKF
Tennis	10	29.14	30.76	**32.05**	31.15
Salesman	10	32.13	34.11	34.41	**35.27**
Coastguard	20	26.62	27.42	**28.40**	27.82
Football	20	24.95	26.62	27.06	**27.78**

Data for 3D KNN, SEQWT and IFSM were extracted from [33]

Table 2 SSIM results compared to other algorithms

Test sequence	Noise	WRSTF [42]	3DWTF [36]	IFSM [33]	STMKF
Foreman	10	**0.914**	NA	0.886	0.885
	15	**0.877**	NA	0.836	0.829
	20	**0.841**	NA	0.793	0.794
Salesman	10	0.932	0.923	0.904	**0.955**
	15	0.901	0.903	0.851	**0.935**
	20	0.868	0.882	0.801	**0.916**
MissA	10	0.905	NA	0.904	**0.937**
	15	0.877	NA	0.857	**0.896**
	20	0.846	NA	0.812	**0.854**
Garden	10	**0.953**	0.909	0.927	0.901
	15	**0.922**	0.872	0.882	0.857
	20	**0.889**	0.840	0.837	0.804

Data for WRSTF, 3DWTF and IFSM were extracted from [13]

decent results on videos that feature rapidly moving objects. In some cases, however, depending on the value of Q and the characteristics of the video, ghosts may be formed in some frames.

6 Performance evaluation

In this section, we evaluate the performance of the parallel implementations of the STMKF algorithm on multicores and GPUs. First, we describe the platforms used in the experiments. Then, we discuss the results obtained with MT-STMKF (multicores) and GPU-STMKF (GPUs).

6.1 Parallel platforms

We evaluate the performance of the parallel implementations of STMKF on four processors. These processors represent three different classes: general purpose (Intel i7), embedded (ARM Cortex-A7) and manycores (GPUs). ARM processors are usually found in embedded systems. In the context of real-time image processing, embedded systems have become highly relevant, with several applications including portable devices, smart cameras and robotics [16, 40]. Table 3 presents the main specifications of each one of the processors considered in this paper.

Since the Intel i7 processor features *Hyper-Threading Technology* (HT), we performed experiments with up to 8 threads (one for each virtual core). For the ARM Cortex-A7 processors, however, we limited our experiments to the number of physical cores (4 cores). For the GPUs, we always use all cores available. All results presented in the next sections are computed based on the average times measured in 30 runs and present statistical confidence of 95 % by Student's t distribution and a maximum of 0.7 % of relative error.

6.2 Multicore results

We analyze the performance of MT-STMKF on the two multicore processors presented in Sect. 6.1 (Intel i7 and ARM Cortex-A7). We use two metrics to evaluate its performance: *frame rate* and *speedup*.

The *frame rate* metric is the frequency at which our algorithm computes consecutive frames. It is expressed in frames per second (FPS). The *speedup* metric, on the other hand, represents the ratio between the execution times of the sequencial solution and the multithreaded solution with a specific number of threads. It is important to notice that the potential speedup gained by a parallel solution is limited by the portion of the sequencial solution that can be effectively parallelized. In the best case, i.e., when 100 % of the sequential solution is parallelized, the speedup

Table 3 Parallel platforms specifications

Type	Model	Cores	Clock	Memory	OS
CPU	Intel Core i7-4710MQ	4 + HT	2.5 GHz	16 GB	Ubuntu 15.04
	ARM Cortex-A7	4	900 MHz	1 GB	Raspbian 7
GPU	NVIDIA Tesla K40	2, 880	745 MHz	12 GB	Ubuntu 14.04.3
	NVIDIA GeForce GTX 850M	640	876 MHz	2 GB	Ubuntu 15.04

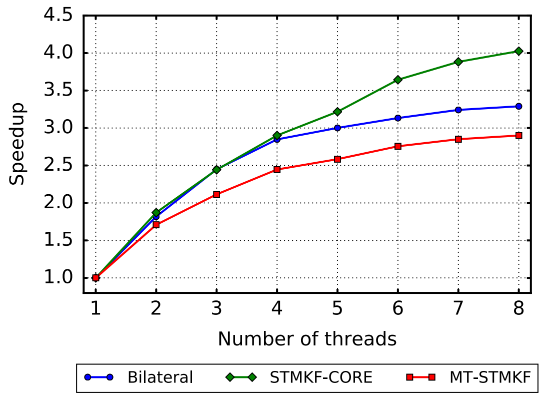

Fig. 6 Scalability of MT-STMKF versus the parallel versions of the Bilateral filter and STMKF-CORE with the 4 K resolution on the Intel i7 processor

increases linearly with the number of physical cores. However, linear scalability is very difficult to achieve since most of the parallel solutions include serial (sequential) portions of code.

Figure 6 presents the speedup of the parallel versions of the Bilateral filter and STMKF-CORE when denoising a 4 K video on the Intel i7 processor. In addition, it presents the overall speedup of MT-STMKF. As it can be observed, the Bilateral filter, which was parallelized by OpenCV, presented worse scalability than our parallel implementation of STMKF-CORE. The MT-STMKF, which includes the parallel versions of Bilateral and STMKF-CORE, presented a slightly worse scalability than parallel Bilateral and STMKF-CORE individually. This behavior is expected and the main reasons are threefold: (1) MT-STMKF has some data dependencies (e.g., Bilateral and Blur filters must be computed before STMKF-CORE); (2) there are portions of the MT-STMKF algorithm that could not be parallelized; and (3) the parallel implementation of the Blur filter implemented by OpenCV did not bring any performance gains.

Figure 7 presents the speedup and FPS obtained with MT-STMKF when varying the number of threads and video resolutions on both Intel i7 and ARM Cortex-A7 processors. We did not include the 240p resolution results for Intel i7 due to the low amount of computation required for this processor. Analogously, we did not include the 4 K resolution results for ARM Cortex-A7 due to the high amount of computation required for this processor.

Overall, MT-STMKF presented a significant speedup when increasing the number of threads. MT-STMKF achieved a speedup of up to 2.9× with eight threads on the Intel i7 processor. Similarly, MT-STMKF achieved a speedup of up to 2.3× with four threads on the ARM Cortex-A7 processor.

Although both processors had a similarly good speedup, there is a considerable difference regarding their FPS due to the different purpose of each processor. With Intel i7, MT-STMKF achieved a maximum of 138 FPS for 480p videos and a maximum of 7.8 FPS for 4 K videos. In contrast, ARM Cortex-A7 is more focused on energy-consumption than performance. For this reason, MT-STMKF achieved a maximum of 18 FPS for 240p videos and a maximum of 1.3 FPS for 1080p videos.

6.3 GPU results

We evaluate the performance of the GPU implementation of the STMKF algorithm (GPU-STMKF) on the two NVIDIA GPUs presented in Sect. 6.1 (GeForce GTX and Tesla K40). In general, GPUs are already known to convey high-performance real-time image processing [29, 35]. Thus, we expect significant performance improvements compared to multicores.

Figure 8 presents the results obtained on both GPUs for various video resolutions. As explained in Sect. 4.3, our first version of GPU-STMKF (called No-S in Fig. 8) performs synchronous data transfers between the host and GPU. The drawback of this approach is that the host can only copy the next frame to be computed on the GPU after receiving the previous one from the GPU. Our second version of GPU-STMKF (called S in Fig. 8) improves the first one by overlapping data transfers between the host and GPU with computations on the GPU. To do so, we perform asynchronous data transfers using CUDA streams.

Overall, the GPU-STMKF version with CUDA streams (S) achieves much better performance than the synchronous version (No-S) on both GPUs. Taking the 480p resolution video as an example, the FPS was improved from 279 to 1, 044 on Tesla K40, whereas it was improved from 229 to 398 on GeForce GTX. This means a performance improvement of 3.7× and 1.7× on Tesla K40 and GeForce GTX, respectively. As we increase the video resolution, however, we observed that the performance improvement of GPU-STMKF with CUDA streams decreases.

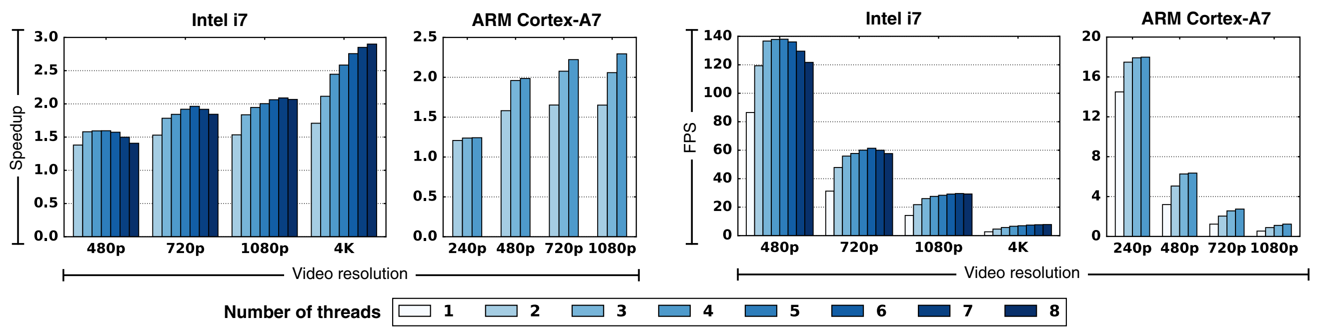

Fig. 7 Speedup and FPS of MT-STMKF on multicore processors when varying the number of threads and frame resolutions

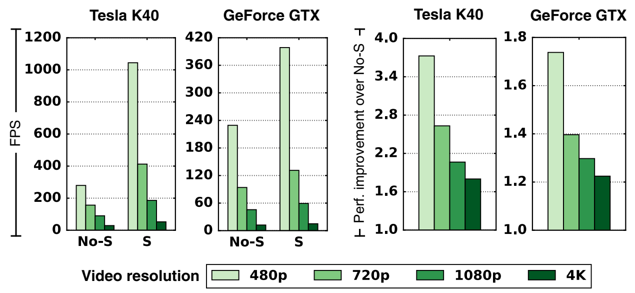

Fig. 8 Performance of GPU-STMKF with (S) and without (No-S) CUDA streams

With the maximum video resolution (4K), we observed a performance improvement of 1.8× and 1.2× on Tesla K40 and GeForce GTX, respectively.

It is worth noting that the performance improvement of GPU-STMKF with CUDA streams is significantly higher on Tesla K40 for all video resolutions tested. The main reason is that Tesla K40 has about 4× more processing power[2] than GeForce GTX. In other words, each frame takes about 4× less time to be computed by GPU-STMKF on Tesla K40 than on GeForce GTX. Because of that, for the synchronous version of GPU-STMKF (No-S), the ratio between the time spent on data transfers between the host/GPU and the time spent on GPU computation is much higher on Tesla K40 than on GeForce GTX. Thus, the benefits of asynchronous data transfers with CUDA streams are more significant on Tesla K40.

6.4 STMKF versus other methods

In this section, we compare the performance of STMKF against the other approaches. We evaluated the performance of IFSM on the same CPUs presented in Table 3, since its source code was available on the Internet.[3] Unfortunately, we were unable to perform the same experiments with the other approaches because their source codes were not available. For the SEQWT and WRSTF, we present the results as reported in [42]. Results for the 3D KNN and 3DWTF approaches were omitted due to the lack of such information.

Table 4 presents the execution time (in seconds) to denoise a frame. As it can be observed, STMKF presents substantial faster computation times than the other approaches. This can be highlighted when comparing the results obtained with the sequential version of STMKF on the low-power ARM processor against SEQWT and WRSTF.

7 Conclusion

In the context of video processing, image noise can hardly be avoided. The computational cost of video denoising algorithms must be as low as possible to be suitable for video capture and transmission in real-time applications. In this paper we proposed STMKF, a real-time video denoising algorithm that fuses Kalman-based and Bilateral filters to achieve decent visual results with low computational costs. We presented parallel implementations of STMKF to exploit the parallelism available in modern processors such as general purpose and embedded multicores as well as GPUs.

We evaluate the effectiveness of STMKF using several videos commonly used in the literature, comparing to other real-time denoising algorithms using both the PSNR and SSIM metrics. STMKF achieved the best score for 50 % of the input videos and a good score for the remaining inputs. Overall, STMKF is competitive with several filters found in the literature for both metrics, especially for videos that feature stationary backgrounds, such as in videoconferencing, video surveillance of private areas and video lectures.

We also evaluated the performance of STMKF on different multicore and GPU architectures. The parallel implementations of STMKF could achieve a significant speedup of up to 2.9× with eight threads on the multicore Intel i7 processor and a speedup of up to 2.3× with four threads on

[2] The GPU processing power is usually measured by the number of floating-point operations it can issue per second (FLOPS).

[3] Source code available at http://teacher.buet.ac.bd/mahbubur/resources/TCSVT_prog.rar.

Table 4 Performance of the sequential version of STMKF compared to other approaches

Method	Time	Frame size	Language and platform	GFLOPS per Core
SEQWT [42]	0.814	352 × 288	C++, Athlon64, 2.4GHz	1.15
IFSM	0.061	352 × 240	MATLAB, Intel Core i7-4710MQ, 2.5GHz	3.67
WRSTF [42]	0.866	352 × 288	C++, Athlon64, 2.4GHz	1.15
STMKF	0.1	352 × 288	C++, ARM Cortex-A7, 900MHz	0.093
STMKF	0.004	352 × 288	C++, Intel Core i7-4710MQ, 2.5GHz	3.67

the ARM Cortex-A7 processor. For GPUs, we compared two implementations of STMKF, the first with a simple synchronous data transfers between the host and GPU and the second with asynchronous CUDA streams, overlapping data transfers with computations. The second version with asynchronous communications achieved an improvement of up to 3.7× over the simple GPU implementation, processing up to 1,000 FPS for 480p videos and about 50 FPS for 4 K videos.

As future work, we intend to automatize the choice of the Q parameter of the STMKF algorithm. Moreover, we intend to evaluate the performance of STMKF on low-power manycore processors such the Mellanox TILE-Gx72 and the Kalray MPPA-256. We also plan to evaluate the use of variations of KF such as the Extended Kalman Filter (EKF) and the Unscent Kalman Filter (UKF). Finally, new efficient algorithms for transferring real-time media streams through the Internet have been researched due to the ever-increasing number of complex real-time media applications [20]. One of such algorithms that has gained attention recently is the High-Efficiency Video Coding (HEVC). Thus, we intend to study how STMKF could be used along with HEVC to improve the quality of real-time media streams. In this context, highly relevant issues that affect real-time video transmission could also be considered such as security [26] and wireless video delivery issues [21, 31].

References

1. Bardu, T.: Variational image denoising approach with diffusion porous media flow. Abstr. Appl. Anal. **2013**, 8 (2013)
2. Buades, A., Coll, B., Morel, J.-M.: Nonlocal image and movie denoising. Int. J. Comput. Vision **76**(2), 123–139 (2008)
3. Buades, A., Coll, B., Morel, J.-M.: A non-local algorithm for image denoising. In: Conference on Computer Vision and Pattern Recognition, CVPR '05, pp. 60–65. Washington, DC, USA, IEEE Computer Society (2005)
4. Chan, T.-W., Au, O.C., Chong, T.-S., Chau, W.-S.: A novel content-adaptive video denoising filter. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 649–652, Philadelphia, USA (2005)
5. Chaudhury, K.N.: Acceleration of the shiftable algorithm for bilateral filtering and nonlocal means. IEEE Trans. Image Process. **22**(4), 1291–1300 (2013)
6. Chen, T.-Y., Chen, T.-H., Su, C.-P., Chen, Y.-J.: The study on video enhancement in the low-light environment by spatio-temporal filtering. In: International Conference on Intelligent Systems Design and Applications (ISDA), vol. 3, pp. 561–564, Kaohsiung, Taiwan (2008)
7. Chenglin Z., Yu, L., Xin, T., Wei, W., Maojun, Z. (2013) Video denoising based on a spatiotemporal Kalman-bilateral mixture model. Sci. World J. 2013 (2013)
8. Dabov, K., Foi, A., Egiazarian, K.: Video denoising by sparse 3D transform-domain collaborative filtering. In: European Signal Processing Conference, pp. 145–149, Poznan, Poland. IEEE (2007)
9. Davis, L., Rosenfeld, A.: Noise cleaning by iterated cleaning. IEEE Trans. Syst. Man Cybern. SMC **8**(9), 705–710 (1978)
10. Dufaux, F., Callet, P.L., Mantiuk, R., Mrak, M.: High Dynamic Range Video: From Acquisition, to Display and Applications. Elsevier (2016). ISBN 9780128030394
11. Farooque, M.A., Sohankar, J.S.: Survey on various noises and techniques for denoising the color image. Int. J. Appl. Innov. Eng. Manage. (IJAIEM) **2**, 217 (2013)
12. Garg, R., Kumar, A.: Comparision of various noise removals using bayesian framework. Int. J. Mod. Eng. Res. (IJMER) **2**, 265 (2012)
13. Han, Y., Chen, R.: Efficient video denoising based on dynamic nonlocal means. Image Vision Comput. **30**, 78–85 (2012)
14. Hong-Zhi, W., Ling, C., Shu-Liang, X.: Improved video denoising algorithm based on spatial-temporal combination. In: International Conference on Image and Graphics (ICIG), pp. 64–67, Qingdao, China. IEEE (2013)
15. Jojy, C., Nair, M.S., Subrahmanyam, G.R.K.S., Raji, R.: Discontinuity adaptive non-local means with importance sampling unscented Kalman filter for de-speckling SAR images. IEEE J. Sel. Top. Appl. Earth Obser. Remote Sens. **6**(4), 1964–1970 (2013)
16. Jung, B., Sukhatme, G.S.: Detecting moving objects using a single camera on a mobile robot in an outdoor environment. In: International Conference on Intelligent Autonomous Systems, pp. 980–987 (2004)
17. Kalman, R.E.: A new approach to linear filtering and prediction problems. Trans. ASME J. Basic Eng. **82**(D), 35–45 (1960)
18. Karnati, V., Uliyar, M., Dey, S.: Fast non-local algorithm for image denoising. In: International Conference on Image Processing (ICIP), pp. 3873–3876. IEEE (Nov 2009)
19. Kirk, D.B., Wen-mei W.H.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann Publishers Inc., San Francisco, 1st edn. (2010). ISBN 0123814723
20. Kokkonis, G., Psannis, K.E., Roumeliotis, M., Ishibashi, Y.: Efficient Algorithm for transferring a real-time HEVC stream with haptic data through the internet. J. Real-Time Image Process. pp. 1–13, (2015). ISSN 1861-8219. doi:10.1007/s11554-015-0505-7
21. Kokkonis, G., Psannis, K.E., Roumeliotis, M., Schonfeld, D.: Real-time wireless multisensory smart surveillance with 3D-HEVC streams for internet-of-things (iot). J. Supercomput. pp 1–19, (2016). ISSN 1573-0484. doi:10.1007/s11227-016-1769-9
22. Kostadin D., Alessandro F., Vladimir K., Karen E.: Image denoising with block-matching and 3D filtering. In: SPIE-IS&T Electronic Imaging, p. 6064 (2006)

23. Li, W., Zhang, J., Dai, Q.: Video denoising using shape-adaptive sparse representation over similar spatio-temporal patches. Signal Proc.: Image. Communication **26**(4–5), 250–265 (2011)
24. Li, X., Zheng, Y.: Patch-based video proc.: a variational bayesian approach. IEEE Trans. Circuits Syst Video Technol **19**(1), 27–40 (2009)
25. Mahmoud, R.O., Faheem, M.T., Sarhan, A.: Intelligent denoising technique for spatial video denoising for real-time applications. In: International Conference on Computer Engineering Systems (ICCES), pp. 407–412, Cairo, Egypt. IEEE (2008)
26. Mahmoudi, M., Sapiro, G.: Fast image and video denoising via nonlocal means of similar neighborhoods. IEEE Signal Process. Lett. **12**(12), 839–842 (2005)
27. Memos, V.A., Psannis, K.E.: Encryption algorithm for efficient transmission of hevc media. J. Real-Time Image Process. pp. 1–10, (2015). ISSN 1861-8219. doi:10.1007/s11554-015-0509-3
28. Mitchell, H.B., Mashkit, N.: Noise smoothing by a fast k-nearest neighbour algorithm. Signal Process. Image Commun. **4**(3), 227–232 (1992)
29. OpenMP Architecture Review Board. OpenMP application program interface version 4.0, July 2013. URL http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
30. Pauwels, K., Tomasi, M., Alonso, J. Diaz., Ros, E., Van Hulle, M. M.: A comparison of fpga and GPU for real-time phase-based optical flow, stereo, and local image features. IEEE Trans. Comput. **61**(7): 999–1012, (2012). ISSN 0018-9340
31. Pizurica, A., Zlokolica, V., Philips, W.: Noise reduction in video sequences using wavelet-domain and temporal filtering. In: Photonics Technologies for Robotics, Automation, and Manufacturing, Int. Soc. for Optics and Photonics, pp. 48–59 (2004)
32. Psannis, K.E.: Hevc in wireless environments. J. Real-Time Image Process. pp. 1–8, (2015). ISSN 1861-8219. doi:10.1007/s11554-015-0514-6
33. Pulli, K., Baksheev, A., Kornyakov, K., Eruhimov, V.: Real-time computer vision with OpenCV. Commun. ACM **55**(6): 61–69, (2012). ISSN 0001-0782
34. Rahman, S.M.M., Ahmad, M.O., Swamy, M.N.S.: Video denoising based on inter-frame statistical modeling of wavelet coefficients. IEEE Trans. Circuits Syst. Video Technol. **17**(2), 187–198 (2007)
35. Ryu, J., Nishimura, T. H.: Fast image blurring using lookup table for real time feature extraction. In: 2009 IEEE International Symposium on Industrial Electronics, pp. 1864–1869 (2009)
36. Seiller, N., Singhal, N., Park, I.K.: Object oriented framework for real-time image processing on GPU. In: International Conference on Image Processing (ICIP), pp. 4477–4480, Hong Kong, China. IEEE (2010)
37. Selesnick, I.W, Li, K.Y.: Video denoising using 2D and 3D dual-tree complex wavelet transforms. In: Annual Meeting on Optical Science and Technology (SPIE), Int. Soc. for Optics and Photonics, pp. 607–618. (2003)
38. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: International Conference on Computer Vision, Bombay, India, pp. 839–846, IEEE (1998)
39. Van De Ville, D., Kocher, M.: SURE-based non-local means. IEEE Signal Process. Lett. **16**(11), 973–976 (2009)
40. Wang, Z., Bovik, A.C., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Proc. **13**(4), 600 (2004)
41. Wolf, W., Ozer, B., Lv, T.: Smart cameras as embedded systems. Computer **35**(9), 48–53 (2002)
42. Zlokolica, V., Pizurica, A., Philips, W.: Wavelet-domain video denoising based on reliability measures. IEEE Trans. Circuits Syst. Video Technol. **16**(8), 993–1007 (2006)
43. Zlokolica, V., Philips, W., Van De Ville, D.: A new non-linear filter for video processing. In: IEEE Benelux Signal Processing Symposium, pp. 221–224 (2002)

Sergio G. Pfleger is a M.Sc. student in Computer Science at the Federal University of Santa Catarina (UFSC), Brazil. He received his B.Sc. degree in Computer Science from the same university in 2013. Currently he is a member and researcher of the Distributed Systems Research Laboratory (LAPESD) at UFSC and CEO of Loeffa Technological Solutions. His research interests include real-time video processing, video denoising, motion estimation, face detection, face recognition and gesture recognition.

Patricia D. M. Plentz is currently an Associate Professor of Computer Science at the Federal University of Santa Catarina (UFSC), Brazil. She received her Ph.D. and M.Sc. from UFSC in 2008 and 2002, respectively, and her B.Sc. degrees in Computer Science from University of Cruz Alta (UNICRUZ) in 2000. Her research interests include real-time systems, forecast of deadline missing and temporal constraints of mobile robots.

Rodrigo C. O. Rocha received the MSc. in Computer Science from the Federal University of Minas Gerais (UFMG), Brazil, in 2015, and the B.Sc. degree in Computer Science from the Pontifical Catholic University of Minas Gerais (PUC Minas), Brazil, in 2012. He is currently an Assistant Professor at PUC Minas for the undergraduate courses of Computer Science and Information Systems. His research interests include optimizing compilers, parallel programming, high-performance computing and machine learning.

Alyson D. Pereira is a MSc. student in Computer Science at the Federal University of Santa Catarina (UFSC), Brazil. He received his B.Sc. degree in Computer Science from the Pontifical Catholic University of Minas Gerais (PUC Minas) in 2015. Currently he is a member and researcher of the Distributed Systems Research Laboratory (LAPESD) at UFSC and the Creative and Parallel Computing Group (Creapar) at PUC Minas. His research interests include parallel programming models, heterogeneous and parallel architectures and machine learning.

Márcio Castro is currently an Associate Professor of Computer Science at the Federal University of Santa Catarina (UFSC), Brazil. He received his Ph.D. from the University of Grenoble Alpes in 2012, his M.Sc. and B.Sc. degrees in Computer Science from PUCRS in 2009 and 2006, respectively. In 2006, he earned an honor distinction (summa cum laude) from PUCRS and the Best Student Award from the Brazilian Computer Society (SBC). His research interests include parallel programming models, thread and data affinity, high-performance parallel applications and parallel architectures (multicores, manycores and accelerators). More information about his current research activities can be found at: www.marciocastro.com.