A New Real-Time Embedded Video Denoising Algorithm

Andrea Petreto, Thomas Romera, Ian Masliah, Boris Gaillard, Manuel Bouyer, Quentin Meunier, Lionel Lacassagne, Florian Lemaitre

A New Real-Time Embedded Video Denoising Algorithm

Andrea Petreto*†, Thomas Romera*†, Florian Lemaitre*, Ian Masliah*, Boris Gaillard†, Manuel Bouyer*, Quentin L. Meunier* and Lionel Lacassagne*

* Sorbonne University, CNRS, LIP6, F-75005 Paris, France. Email: firstname.lastname@lip6.fr

†Lhéritier - Alcen, F-95862, Cergy-Pontoise, France. Email: bgaillard@lheritier-alcen.com

***Abstract*—Many embedded applications rely on video processing or on video visualization. Noisy video is thus a major issue for such applications. However, video denoising requires a lot of computational effort and most of the state-of-the-art algorithms cannot be run in real-time at camera framerate. This article introduces a new real-time video denoising algorithm for embedded platforms called RTE-VD. We first compare its denoising capabilities with other online and offline algorithms. We show that RTE-VD can achieve real-time performance (25 frames per second) for qHD video (960×540 pixels) on embedded CPUs and the output image quality is comparable to state-of-the-art algorithms. In order to reach real-time denoising, we applied several high-level transforms and optimizations (SIMDization, multi-core parallelization, operator fusion and pipelining). We study the relation between computation time and power consumption on several embedded CPUs and show that it is possible to determine different frequency and core configurations in order to minimize either the computation time or the energy.**

***Index Terms*—Embedded system, Video denoising, Real-time, Image processing, night-vision, SIMD.**

I. Introduction

Image denoising is a major research area for computer vision. It can be based on methods such as pixel-wise filters [1]–[3], patch-based methods [4], [5] or Convolutional Neural Networks (CNN) [6]–[8]. Video denoising can take advantage of the temporal dimension to reduce the noise more efficiently.

Denoising algorithms can be classified according to two criteria: the processing time and the denoising efficiency. In this work, we focus on video with heavy noise and we target real-time processing at 25 frames per second (fps) on embedded systems. Concerning the image quality, we focus on details enhancement rather than aesthetic rendering.

Some video denoising algorithms are able to deal with heavy noise [9]–[14]. However, all these algorithms are slow and cannot be used for real-time denoising. Moreover, they have a high latency as they need future frames to process the current one (typically 5 to 7).

There are algorithms designed for real-time denoising [15] on embedded architectures [16], but only for light noise like compression artifacts, which give poor results on heavy noise.

As far as we know, there is no real-time algorithm able to handle heavily noisy video as in this work.

In this paper, we introduce a new Real-Time Embedded Video Denoising (RTE-VD) algorithm targeting very noisy video in section II, which has been implemented for general purpose processors. We then compare our algorithm to the state-of-the-art in terms of denoising efficiency in section III. We then present the main optimizations applied to achieve real-time performance in section IV. Finally in section V, we compare the execution time of RTE-VD to other state-of-the-art algorithms and study its speed and power consumption depending on various embedded CPUs and frequencies.

II. Denoising algorithm

Most state-of-the-art video denoising algorithms rely on patch based filtering methods [10], [13], [14] which tend to be effective to reduce noise but are very time consuming. In addition, their memory access patterns generate many cache misses, making it difficult to speed the computation up. Therefore, we did not consider this technique for real-time denoising.

A crucial aspect of video denoising is to keep temporal coherence between frames. While we cannot ensure this coherence using patch search, it is possible to do so with an optical flow estimation [11], [17].

Optical flow algorithms are also time consuming but are sensitive to code transformations and can be highly accelerated [18]–[22]. We thus decided to use an optical flow algorithm in our real-time denoising chain.

While there exists many optical flow algorithms [23], optical flow estimation for video denoising should be dense, robust to noise and handle flow discontinuities [17]. Therefore, we decided to use the TV-L1 algorithm to compute the optical flow, since it satisfies all these constraints [24].

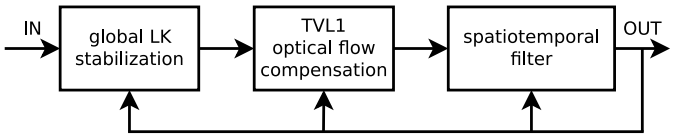

Fig. 1. Principle components of the denoising application.

Our complete denoising application is composed of three steps (Figure 1). First, the video stream is stabilized using a global one-pass Lucas-Kanade approach [25] in order to compensate camera motion. Then, the dense optical flow is computed using the TV-L1 algorithm. The flow enables to extrapolate denoised past frames at the current frame, and match the two together. Finally, a 3D spatio-temporal bilateral filter is applied [26].

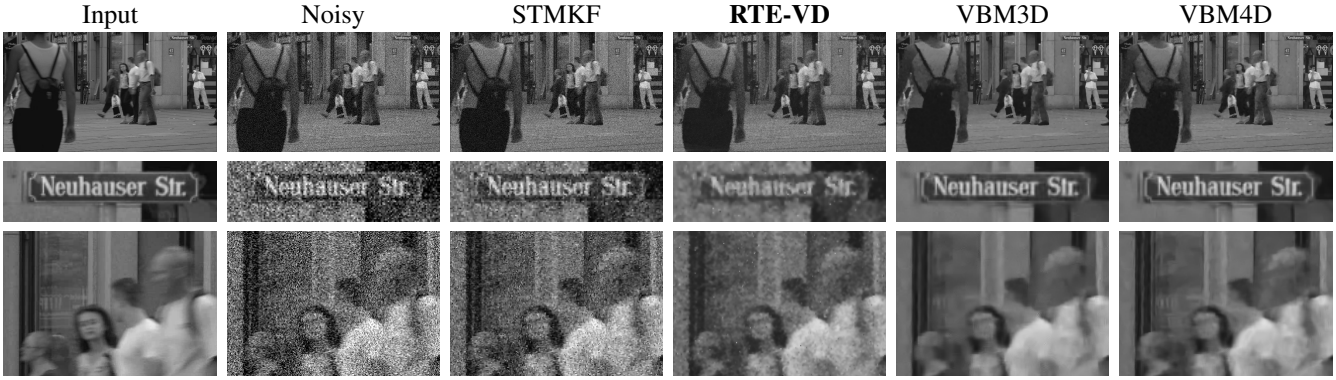

Fig. 2. Visual comparison on pedestrians sequence with a standard deviation noise of 40 (PSNR in Table I).

TABLE I
PSNR RESULTS FOR SEVEN SEQUENCES OF THE *Derf's Test Media Collection* ON STATE-OF-THE-ART ALGORITHMS AND OUR METHOD (RTE-VD).

Noise	Method	crowd	park_joy	pedestrians	station	sunflower	touchdown	tractor	overall
$\sigma = 20$	STMKF [15]	26.25	25.59	28.34	26.66	26.97	28.87	25.37	26.70
	RTE-VD	26.38	25.65	30.58	30.98	32.51	30.17	29.38	**28.73**
	VBM3D [12]	28.75	27.89	35.49	34.19	35.48	32.85	31.44	31.34
	VBM4D [13]	28.43	27.11	35.91	35.00	35.97	32.73	31.65	31.11
$\sigma = 40$	STMKF [15]	20.80	20.75	20.70	20.41	20.70	20.86	19.80	20.56
	RTE-VD	22.55	21.64	25.72	27.76	27.87	27.05	25.99	**24.85**
	VBM3D [12]	24.81	23.78	30.65	30.62	30.88	30.21	27.82	27.43
	VBM4D [13]	24.65	23.22	31.32	31.53	31.39	30.09	28.09	27.35

III. DENOISING EFFICIENCY COMPARISON

In order to evaluate the denoising efficiency of RTE-VD, we reproduced the conditions of the comparison made in [27]. We ran our algorithm on seven sequences from the *Derf's Test Media Collection* [28]. The sequences are originally colored 1080p video. We down-scaled them to 960×540 pixels in gray-scale by keeping only brightness. A Gaussian noise is added to each pixel with a standard deviation of 20 and 40. Then, the algorithms are ran on the first hundred frames of each sequence.

However, we were not able to reproduce the same results as in [27]: all the PSNR values that we have obtained are significantly lower for a given method.

We compare RTE-VD to VBM3D [12], VBM4D [13] and STMFK [15]. VBM3D is one of the most popular video denoising method. It is also one of the fastest high performance denoising algorithm even if its computation time is still over a second for a 960×540 pixel frame on a Xeon server. VBM4D has a more powerful filtering than VBM3D but at a higher computational cost. STMKF is a state-of-the-art real-time video denoising method. For each algorithm, the source code provided on the authors website is used with the default parameters depending on the noise intensity.

The quantitative results for the test sequences are presented in Table I. Our real-time constraint limits the denoising efficiency of RTE-VD compared to VBM3D and VBM4D; however, RTE-VD outperforms STMKF. RTE-VD and STMKF are able to perform real-time denoising while VBM3D and VMB4D require offline computation. Processing times are studied in more details further in this article.

Some visual results are presented in Figure 2: we can see that RTE-VD is a clear improvement compared to the noisy input. In most cases, VBM3D and VBM4D are visually better. Also STMKF is much less effective than the other considered methods.

For the two noise deviation tested, the RTE-VD PSNR is overall less than 3 dB below the ones from VBM3D and VBM4D. RTE-VD PSNR is overall 2 dB above STMKF for a noise deviation of 20 and more than 4 dB above for a noise deviation of 40. It confirms that RTE-VD handles heavy noisy situations better than STMKF, as can be seen in Figure 2.

RTE-VD tends to trade noise reduction off for information keeping. Figure 2 shows that RTE-VD is able to produce more detailed results than STMKF on moving objects but encounters difficulties on still objects. The loss of sharpness in static scenes can be explained by a bad optical flow estimation on non-moving objects. Error estimations are indeed proportionally more important on small movements and bad compensations may generate a little blur effect on non moving objects. Filtering the estimated optical flow may reduce this effect and will be considered in our futur works.

IV. ALGORITHM OPTIMIZATIONS

In order to run RTE-VD in real-time on an embedded system, several optimizations and tradeoffs have to be done.

We apply the following high-level transformations (described in [29]) to each part of the chain:

- SIMDization (handcrafted Neon instructions as the compiler vectorization is inefficient),
- Multi-thread parallelization (with OpenMP),
- Operator fusion to reduce memory accesses,
- Operator pipeline to increase memory access locality,
- Modular memory allocation (for cache blocking) to reduce the memory footprint and enforce memory access locality.

A. Global Lucas-Kanade stabilization

To stabilize the video stream, we use a modified version of the Lucas-Kanade method [25] with a global approach. the input image is convolved with a gate function, (a function that is 0 outside a specified interval and 1 inside it) in order to compute its average over a neighborhood. As the gate size depends on the largest movement to compensate, the convolution kernels can be very large. To speed the convolution up, integral images (also known as *summed area table* [30]) are used. The convolution is computed by multiple threads, each one processing a strip. Consequently, each thread only computes the partial integral image it needs. We then apply the Lucas-Kanade flow estimation to a neighborhood of the same size as the input images.

B. TV-L1 Dense Optical Flow Estimation

The TV-L1 optical flow algorithm is the main step to optimize as it takes more than 80% of the total time. The key transformations of TV-L1 are presented in [18] and result in a processing $5\times$ faster and $6\times$ less power consuming on an embedded ARM Cortex A57 architecture.

With n scales and a zoom factor of $1{:}\,2$, the largest movement between two images that the algorithm can handle is equal to $2^n - 1$. Therefore we fixed the scale number to 3 to be able to estimate displacements as large as 7 pixels.

Instead of using a convergence criteria to stop the iterations, a fixed number of iterations per scale has been chosen. In most situations, three iterations per scale are enough. However, more complex situations, like large displacements or discontinuities, require more iterations. Therefore, we set 10 iterations per scale as the standard configuration. This provides a more robust estimation than with only three iterations per scale, while keeping a rather low computation time.

However, as the most important movements to compensate are the largest ones, it is interesting to do more iterations on the most zoomed-out scales (the smallest ones). Therefore, we compare the computation of 10 iterations per scale on 3 scales (10-10-10) with the computation of 3 iterations at the largest scale, 20 iterations at the medium scale and 80 iterations at the smallest one (3-20-80). While the two have roughly the same number of operations, the (3-20-80) version is actually faster as it spends more time on the smallest scale which is cache friendlier. The denoising quality is similar.

Another critical step of TV-L1 to ensure stability is the *warping*, which is in our case a motion compensation using bicubic interpolation. Several warps can be applied at each scale. They take a long time to compute, but gives better results. This leads to important design choices, in particular regarding the number of warps at each scale. Thus, we compared the (10-10-10) single warp per scale configuration with a configuration comprising 3 iterations with 1 warp at the largest scale, 20 iterations with 2 warps at the medium scale and 80 iterations with 4 warps at the smallest scale. This latter configuration is the one used in the remaining experiments, as it significantly achieves better results with a similar computation time for 960×540 pixel images.

C. Spatio-temporal filter

To actually filter out the noise, we compose a spatial bilateral filter [26] with a unilateral filter to handle the temporal dimension and speed the processing up. Thus, we are able to decorrelate the strength of the filter in the spatial domain from the temporal domain. The filter is defined by the equations 1, 2 and 3, where I_f is the filtered image, I_p the previous compensated and filtered image, I the current image and x the coordinates of the current pixel. Ω is the filter kernel domain. σ_i, σ_d and σ_t are the smoothing parameters of the filter respectively of the spatial intensity difference, the distance and the temporal intensity difference between pixels.

$$I_f(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I_t(x_i) e^{\frac{-[I_t(x_i) - I_t(x)]^2}{2\,\sigma_i^2}} \times e^{\frac{-[x_i - x]^2}{2\,\sigma_d^2}} \tag{1}$$

Where:

$$W_p = \sum_{x_i \in \Omega} e^{\frac{-[I_t(x_i) - I_t(x)]^2}{2\,\sigma_i^2}} \times e^{\frac{-[x_i - x]^2}{2\,\sigma_d^2}} \tag{2}$$

and:

$$I_t(x) = I_p(x) e^{\frac{-[I_p(x) - I(x)]^2}{2\,\sigma_t^2}} + I(x) \left(1 - e^{\frac{-[I_p(x) - I(x)]^2}{2\,\sigma_t^2}}\right) \tag{3}$$

While the bilateral filter is not separable, we approximate it by a separable filter [31] which is faster than the exact one. We also used a fast approximation of the exponential function which manipulates the bit representation of floats as defined in the IEEE-754 standard. Such an approximation is described in [32] and is accurate enough.

V. Processing time and energy consumption

In this section we first compare the execution time of RTE-VD to the algorithms previously introduced. We then briefly study the impact of our optimizations on computation speed. Finally, we evaluate the performances of RTE-VD on various embedded CPUs and frequencies.

We consider 4 different platforms. The first one is an Intel Xeon Silver 4114 $2\times$10C/20T@2.20GHz. The 3 others are the latest Nvidia Jetson embedded platforms; for those platforms, we only consider the ARM CPUs and not the GPUs. Their names and specifications are given in Table II.

TABLE II
TECHNICAL SPECIFICATIONS OF THE TARGET EMBEDDED BOARDS.

Board	Process	CPU	Fmax (GHz)	Idle Power (W)
TX2	16 nm	4×A57 + 2×Denver2	2.00	2.0
AGX	12 nm	8×Carmel	2.27	6.3
NANO	12 nm	4×A57	1.43	1.2

TABLE III
DENOISING TIME DEPENDING ON THE USED METHOD AND THE TESTED PLATFORM FOR 960×540 PIXELS IMAGES.

Algorithm	Time (s)	Platform
STMKF [15]	0.004 5	Xeon
RTE-VD (this work)	0.009 7	Xeon
VBM3D [12]	2.0	Xeon
VBM4D [13]	45	Xeon
STMKF [15]	0.015	AGX
RTE-VD (this work)	0.037	AGX

A. Processing time analysis

We measured the computation time for the different methods considered in section III. We ran RTE-VD, STMKF, VBM3D and VBM4D on the Xeon platform. We also tested RTE-VD and STMKF on the AGX platform. The results for 960×540 pixel images are presented in Table III. On the same platform, RTE-VD is more than 200× faster than VBM3D and more than 4600× faster than VBM4D. On the AGX platform, RTE-VD is only 2.5× slower than STMKF but still achieves real-time processing with 26.7 frames per second.

To exhibit the efficiency of our optimizations we also compare our *Fast* optimized implementation to a *Slow* straightforward implementation. The integral image is hard to parallelize without using major transformations. Those transformations have only been applied to the *Fast* version. Thus, in order to be fair, we compare the *Fast* and *Slow* versions in mono-thread and then analyze only the *Fast* version in multi-thread. The images used for the experiments are square images, with a width varying from 200 to 1500 pixels. The size of the image has negligible impact on the speed of the processing chain. Thus, 1000×1000 images are considered for the following experiments.

The results for all algorithms involved in RTE-VD on the AGX are presented in Table IV. It shows that in mono-thread, we have an overall speedup of ×18, and in multi-thread, a speedup of ×70 using the 8 cores of the AGX. We can also observe that the major part of the speedup is obtained on the filtering, mainly due to its approximation with a separable filter. As a consequence, the optical flow estimation now takes almost 98% of the total time of the *Fast* version, while filtering was the most time consuming part of the processing chain without the optimizations. Since the optical flow estimation is, in the end, the most consuming step, we produced a more detailed study in [18]. In this previous work the impact of each optimisation is discussed. Both time and power consumption are considered.

TABLE IV
EXECUTION TIME (MS) AND SPEEDUP OF RTE-VD ON AGX CPU

Algorithm	Slow 1C	Fast 1C	Fast 8C	speedup
LK	6.66	1.59	0.37	×18
Flow	260.73	107.93	27.59	×10
Filter	1 717.85	1.39	0.25	×6 871
Total	1 985.24	110.90	28.21	×**70**

B. Time and energy efficiency of RTE-VD

Since we target embedded systems, we have to consider not only the computation speed but also the energy consumption. Therefore, we have run RTE-VD at various frequencies on the three Nvidia embedded systems. The frequencies have been taken among the available frequencies of each board and the external memory frequency has been set to its maximum. For the AGX and NANO, we have used a multi-threaded version on all the physical cores, respectively 8 and 4. For the TX2, we have first used only the two Denver cores, then only the four A57 cores and finally all 6 cores. The cooling system of each target board has been set to the maximum and the energy saving policies of the Operating System have been deactivated. We have simultaneously measured time and power consumption for various frequencies and image sizes.

In order to perform simple and reproducible power measurements, the electrical consumption of the entire system has been measured. A board was developed to this effect and has been inserted between the power source and the target system. The board samples both voltage and current at 5 kHz. Measurements and code executions are synchronized using GPIOs.

We can define four different metrics related to power consumption:

- *Static energy*: the energy associated to the static power when the system is idle (here, when the power consumption comes from the leakage and running the operating system)
- *Dynamic energy*: the energy associated to the dynamic power: that is the extra energy consumed by the computation: the total energy minus the static energy.
- *Compute energy*: the energy consumed by the whole system, that is the sum of static and dynamic energy.
- *Period energy*: the energy consumed by the whole system between two execution starts, including the waiting time.

The period energy is more relevant if we consider a complete system for which we know all of its applications and behaviours. Since this is not yet the case here, we only consider compute and dynamic energy in the following.

Our measurements show that considering the compute energy, the maximum frequency is always the most efficient for all configurations, being both the fastest and the least

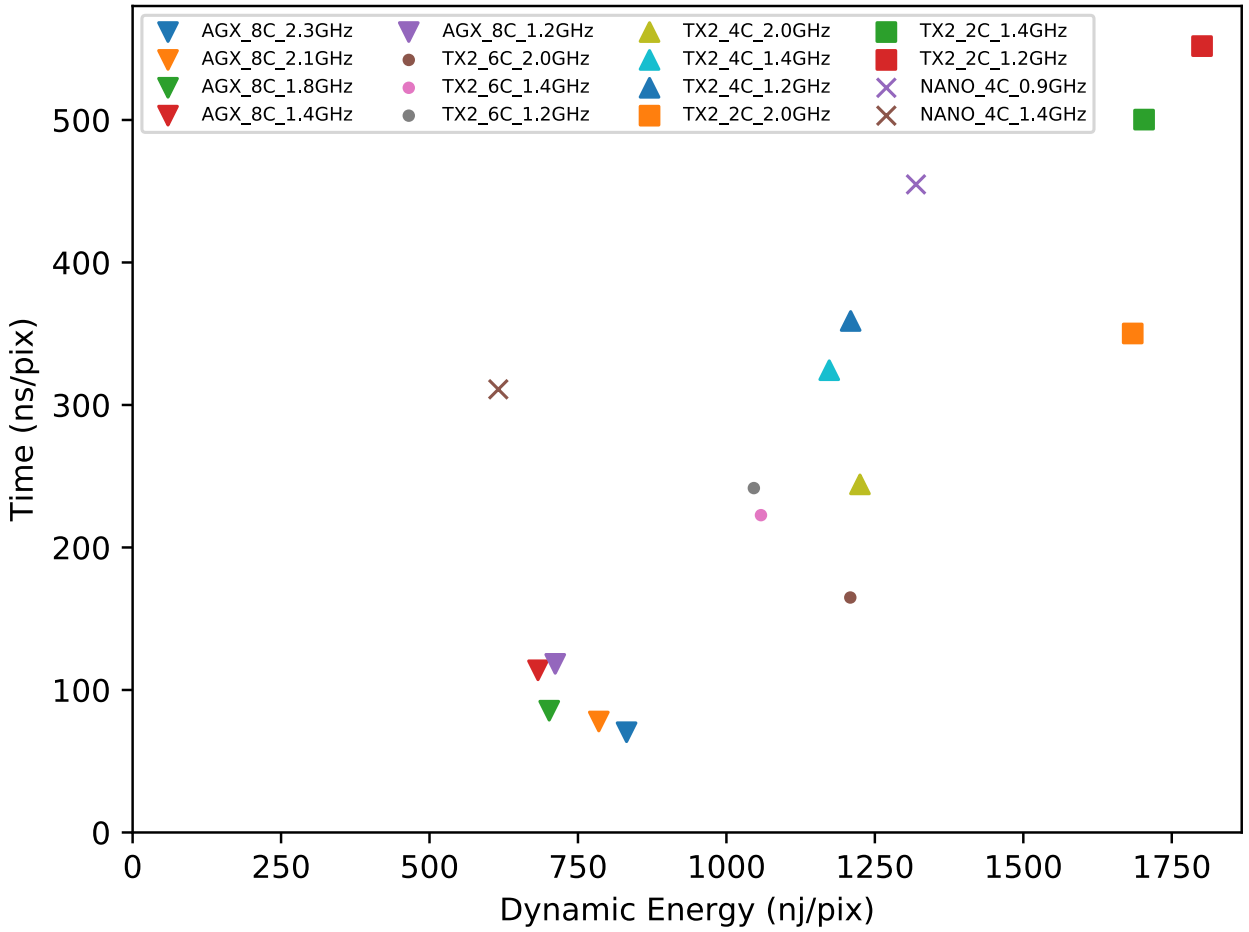

Fig. 3. Speed and energy efficiency of RTE-VD for embedded CPUs at different frequencies

energy consuming. Considering dynamic energy, results show that there is a possible tradeoff between speed and energy consumption by selecting different frequencies values.

Figure 3 represents the computation time in nanosecond per pixel and the dynamic energy consumption in nanojoule per pixel for all the boards and frequencies. The NANO is the least energy consuming; however, it is possible to be 2.7× faster while consuming almost as little energy using the AGX. The AGX is the fastest since it is 2.3× faster than the fastest configuration of the TX2 and 4.4× faster than the fastest NANO configuration. Due to its etching process (16nm), the TX2 is often less energy efficient and slower than the two others (12nm). This is especially visible if we compare the NANO to the TX2 using only its A57 quad core at equivalent frequency. Since the NANO also possesses a quad core A57, it is able to compute as fast as the TX2 while consuming 1.6× less. However the maximum frequency of the TX2 being higher, it is able to be faster than the NANO even using only the A57 cores.

The Table V synthesizes the best configurations minimizing either energy consumption or computation time for each platform. It gives for each configuration the largest image size possibly processed at 25 frames per second.

TABLE V
BEST CONFIGURATIONS FOR REAL-TIME DENOISING AT 25 FPS

Configuration	Energy (nJ/pix)	Time (ns/pix)	Max size (#pix)	Freq (GHz)
NANO min energy NANO min time	616	311	358	1.4
TX2 min energy TX2 min time	1046 1209	242 165	406 492	1.2 2.0
AGX min energy AGX min time	683 832	114 70	592 754	1.4 2.3

VI. CONCLUSION

In this article, we proposed a novel real-time embedded video denoising chain called RTE-VD, which is able to restore details on very noisy video while achieving high performance on embedded systems.

In order to achieve real-time processing, we applied several code transformations (like SIMDization, multi-threading, operator fusion and pipelining) and were able to get an implementation 70× faster than a naive one. The PSNR and visual results from our experiments validates our approach.

We thus compared RTE-VD to other state-of-the-art algorithms: VBM3D, VBM4D and STMKF. RTE-VD always denoises better than STMKF while being less effective than the more costly algorithms VBM3D and VBM4D. On heavy noise situations ($\sigma = 40$), RTE-VD achieves an overall PSNR more than 4 dB above STMKF. RTE-VD is also $200\times$ faster than VBM3D and $4600\times$ faster than VBM4D.

While RTE-VD is $2.5\times$ slower than STMKF, it is still able to process 960×540 pixel video at 25 fps on a Nvidia Tegra AGX. Given these results, we believe that RTE-VD is a denoising algorithm particularly well positioned for speed/accuracy tradeoff.

Since we are targeting embedded systems, we also studied the link between time and energy consumption on various embedded CPUs. Within the tested platforms, it appears that the Nvidia Jetson Nano is the least power consuming platform while the Nvidia Tegra AGX is the fastest. We were also able to determine for each platform multiple efficient frequencies minimizing either computation time or energy consumption.

As future work, we plan to increase the whole performance by balancing the load on the CPU and the GPU and by studying what parts of the algorithm can be hybridized with 32- and 16-bit computation.

ACKNOWLEDGMENT

The authors would like to thank the French *Direction Générale de l'Armement* for supporting this work.

REFERENCES

[1] M. Zhang and B. K. Gunturk, "Multiresolution bilateral filtering for image denoising," *IEEE Transactions on image processing*, vol. 17, no. 12, pp. 2324–2333, 2008.

[2] T. Chen, K.-K. Ma, and L.-H. Chen, "Tri-state median filter for image denoising," *IEEE Transactions on Image processing*, vol. 8, no. 12, pp. 1834–1838, 1999.

[3] A. Buades, B. Coll, and J.-M. Morel, "A review of image denoising algorithms, with a new one," *Multiscale Modeling & Simulation*, vol. 4, no. 2, pp. 490–530, 2005.

[4] ——, "A non-local algorithm for image denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 60–65.

[5] M. Lebrun, A. Buades, and J.-M. Morel, "A nonlocal bayesian image denoising algorithm," *SIAM Journal on Imaging Sciences*, vol. 6, no. 3, pp. 1665–1688, 2013.

[6] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain neural networks compete with bm3d?" in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 2392–2399.

[7] V. Jain and S. Seung, "Natural image denoising with convolutional networks," in *Advances in neural information processing systems*, 2009, pp. 769–776.

[8] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[9] C. Zuo, Y. Liu, X. Tan, W. Wang, and M. Zhang, "Video denoising based on a spatiotemporal kalman-bilateral mixture model," *The Scientific World Journal*, vol. 2013, 2013.

[10] P. Arias, G. Facciolo, and J.-M. Morel, "A comparison of patch-based models in video denoising," in *2018 IEEE 13th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*. IEEE, 2018, pp. 1–5.

[11] A. Buades and J.-L. Lisani, "Video denoising with optical flow estimation," *Image Processing On Line*, vol. 8, pp. 142–166, 2018.

[12] K. Dabov, A. Foi, and K. Egiazarian, "Video denoising by sparse 3d transform-domain collaborative filtering," in *2007 15th European Signal Processing Conference*. IEEE, 2007, pp. 145–149.

[13] M. Maggioni, G. Boracchi, A. Foi, and K. Egiazarian, "Video denoising using separable 4d nonlocal spatiotemporal transforms," in *Image Processing: Algorithms and Systems IX*, vol. 7870. International Society for Optics and Photonics, 2011, p. 787003.

[14] P. Arias and J.-M. Morel, "Towards a bayesian video denoising method," in *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, 2015, pp. 107–117.

[15] S. G. Pfleger, P. D. Plentz, R. C. Rocha, A. D. Pereira, and M. Castro, "Real-time video denoising on multicores and gpus with kalman-based and bilateral filters fusion," *Journal of Real-Time Image Processing*, pp. 1–14, 2017.

[16] J. Ehmann, L.-C. Chu, S.-F. Tsai, and C.-K. Liang, "Real-time video denoising on mobile phones," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 505–509.

[17] C. Liu and W. T. Freeman, "A high-quality video denoising algorithm based on reliable motion estimation," in *European Conference on Computer Vision*. Springer, 2010, pp. 706–719.

[18] A. Petreto, A. Hennequin, T. Koehler, T. Romera, Y. Fargeix, B. Gaillard, M. Bouyer, Q. L. Meunier, and L. Lacassagne, "Energy and execution time comparison of optical flow algorithms on SIMD and GPU architectures," in *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE, 2018, pp. 25–30.

[19] T. Kroeger, R. Timofte, D. Dai, and L. V. Gool, "Fast optical flow using dense inverse search," in *(ECCV)*, 2016.

[20] A. Plyer, G. L. Besnerais, and F. Champagnat, "Massively parallel lucas kanade optical flow for real-time video processing applications," *Journal of Real-Time Image Processing*, vol. 11,4, pp. 713–730, 2016.

[21] M. Kunz, A. Ostrowski, and P. Zipf, "An FPGA-optimized architecture of horn and schunck optical flow algorithm for real-time applications," in *International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–4.

[22] L. Bako, S. Hajdu, S.-T. Brassai, F. Morgan, and C. Enachescu, "Embedded implementation of a real-time motion estimation method in video sequences," *Procedia Technology*, vol. 22, pp. 897–904, 2016.

[23] Middlebury, "Optical flow database http://vision.middlebury.edu/flow/."

[24] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l 1 optical flow," in *Joint Pattern Recognition Symposium*. Springer, 2007, pp. 214–223.

[25] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[26] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *null*. IEEE, 1998, p. 839.

[27] A. Davy, T. Ehret, G. Facciolo, J.-M. Morel, and P. Arias, "Non-local video denoising by cnn," *arXiv preprint arXiv:1811.12758*, 2018.

[28] Derf, "Derf's test media collection https://media.xiph.org/video/derf/."

[29] L. Lacassagne, D. Etiemble, A. Hassan-Zahraee, A. Dominguez, and P. Vezolle, "High level transforms for SIMD and low-level computer vision algorithms," in *ACM Workshop on Programming Models for SIMD/Vector Processing (PPoPP)*, 2014, pp. 49–56.

[30] F. C. Crow, "Summed-area tables for texture mapping," in *ACM SIGGRAPH computer graphics*, vol. 18, no. 3. ACM, 1984, pp. 207–212.

[31] T. Q. Pham and L. J. Van Vliet, "Separable bilateral filtering for fast video preprocessing," in *2005 IEEE International Conference on Multimedia and Expo*. IEEE, 2005, pp. 4–pp.

[32] N. N. Schraudolph, "A fast, compact approximation of the exponential function," *Neural Computation*, vol. 11, no. 4, pp. 853–862, 1999.