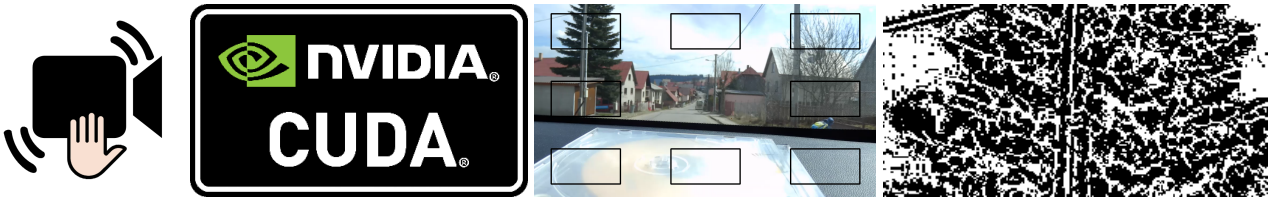
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

CUDA Accelerated Real-time Digital Image Stabilization in a Video Stream

Dávid Pacura*



Abstract
The most important step for successful video processing in computer vision is its stabilization. Often, it is required to process high resolution video in a real-time. In this paper, a new method for a real-time digital image stabilization in a video stream is presented. This method preserves the intended camera motion and exploits computing power of GPGPU by utilizing CUDA programing interface. In order to reduce required computation power, local search windows are used for the correspondence search of consecutive video frames. These windows are further processed using Local Binary Patterns, which enables fast correlation using bitwise XOR. The experiments on video sequences from both car-mounted and hand-held camera have demonstrated the effectiveness of this method. The speed of stabilization designates this method for video preprocessing in real-time applications.

Keywords: Digital Image Stabilization — CUDA Video Stabilization — Real-time Video Stabilization

Supplementary Material: Demonstration Video

*xpacur00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Over the past two decades, a rapid development in the field of electronic technology brought, among other things, an increase of use of digital video cameras. Nowadays, as a consequence, a wide variety of use cases where digital video camera is used exists (e.g. video surveillance, reconnaissance, motion detection, target tracing or automatic recognition). However, in many cases, camera device is mounted on moving objects (e.g. ships, vehicles) or on high poles and towers, where object movement or gusting wind causes camera shaking. In a lot of these cases, high resolution, high frames count per second and steady image without parasitic effects like shake, jitter and blur is required. This is due to requirements for successful post-processing like target tracking or movement detection [1].

However, there is often limited space, resources or both to fulfill these requirements – usage of hardware stabilization is in many cases highly restricted or even impossible. Yet, a digital image stabilization can be used. This enables a use of smaller video cameras, but requires high computing power for post-processing. Often, real-time processing of data is also required, which increases requirements for computing power even more, because not only video stabilization itself, but also additional required steps in the process must be resolved almost immediately. This can be achieved by specialized hardware like FPGA (field programmable gate array) or using GPGPU (general purpose graphics processing unit). However, this paper will focus only on GPGPU because of its easy availability, low price and relatively easy programming [2].

The key to video stabilization is the accurate global motion estimation. There are many methods based on various approaches including gray based [3], frequency based [4] or feature based [5] methods. Also, motion estimation can be computed in 2D [6] [7], which is suitable for long monitoring distances in outdoor conditions or 3D [8], which are suitable for low focal distances, where obvious changes in parallaxes inducted by 3D viewpoint translations occurs. However, many of these approaches are not suitable for real-time processing even in case of GPGPU use.

Another important step of video stabilization is motion filtering, when unwanted shaking and jitter must be removed, but intended motion preserved. A number of low pass filtering methods such as fuzzy filtering [9] or Gaussian weighting [10] exist. However, these methods are not suitable for real-time processing, as they require information about previous frames, or in worse case information about future frames, which introduces undesirable delay. Another approaches are use of Kalman filter [11] [12] which is an optimal filter in the minimum variance sense. This filter is fast, but sensitive to parameter values. Therefore, modifications like Adaptive Kalman filter [13] [14] exists. Another example is Particle filter [15] [16] which is suitable for filtering of non-linear motion of the camera. However, it has a great processing time consumption.

In our solution, a HD video is stabilized in real-time using GPU and CUDA. For this, a binary technique is used for global motion estimation. Firstly, eight areas of concern (SSW's – searching sub-windows) are selected in each frame. Then, they are binarized using enhanced Local Binary Patterns [17] method. Next, the correlation template (MSW – matching sub-window) is selected as a central part of corresponding SSW and is correlated with previous SSW using Number of non-matching points (NNMP) [17]. Then, eight best local motion vectors (LVMs) for each SSW is selected in order to enhance the motion estimation. The last step is motion filtering. It is achieved by selection of median value from LVMs. Selected value represents the translation between frames. However, in order to preserve intended camera motion, Kalman filtering of selected translation is performed.

The proposed solution utilizes GPGPU and CUDA to enable real-time stabilization of HD video with the pixel precision even on older hardware. However, it is for the price of no angle compensation between subsequent frames. Another problem is Kalman filtering - it is unable to distinct between high jitter and sudden intended motion, which results into delayed reaction to compensate sudden change of position.

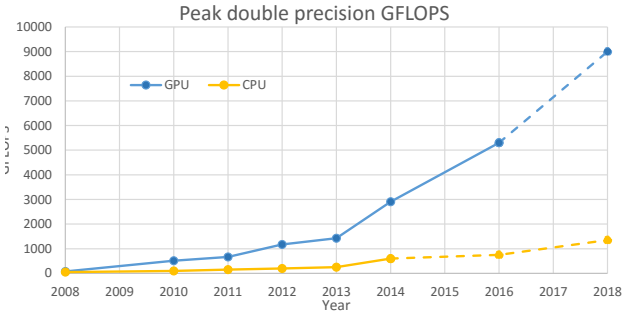


Figure 1. Development and prediction of CPU and GPU computing power.

2. GPGPU acceleration

The general purpose graphic processing units (GPGPUs) are phenomenon of the last decade. Their raw power greatly outperforms those of CPUs due to the use of hundreds of simple computing cores (see Figure 1). Yet, GPU is still an accelerator connected through peripheral component interconnect express bus (PCI-E) and requires host processor (CPU) for work scheduling.

Further, the use of GPU brings multiple issues – beside slow access to GPU memory (about 700 clock cycles), it is also branching sensitivity. This is a hardware limitation (compromise between speed and universality). For this reason, executing threads are organized into groups of 32, called warps [18]. As a result, threads in the same warp should all take the same branch, otherwise performance penalty will occur – threads execution will be serialized [18] [19].

Therefore, the video stabilization is an ideal task for GPGPU, as the same set of operations can be performed in parallel on all pixels of image. However, in order to provide huge performance speedup over CPU, the algoithm must be properly designed. Otherwise, it may not be faster than CPU.

2.1 CUDA framework

The CUDA is a proprietary framework developed by NVIDIA which specializes in graphics cards development. It works only with NVIDIA GPUs. Main features of current version 7.5 are: unified memory between host and device, libraries for GPU code, new work spawning from within GPU code or C++11 features like lambdas or auto type specifiers. All CUDA versions are backward compatible, forward compatibility is guaranteed on binary level. Yet, the new features are often limited to new hardware [20] [21] [22] [23].

Further, a great collection of powerful tools and libraries exists [24]: advanced debugger and profiler or official free libraries (e.g. image processing primitives, Standard Template Library equivalent).

3. Proposed method for GPGPU video stabilization

In order to stabilize video, a correspondence between each pair of consecutive frames must be found. The accuracy of this step is crucial to the successful video stabilization. In proposed method, this is achieved using binarization approach. The second, but also important step is motion filtering – the intended camera motion should be preserved, while jitter should be removed. This is achieved by Kalman filtering.

However, before proceeding, the worst case scenario of video that should be successfully stabilized and desired properties must be defined:

- Shaking up to the frequency of 15 Hz.
- FullHD input resolution.
- 30 frames per second.
- Real-time processing.

3.1 Areas of concern selection

Figure 2. Input frame with marked areas of interest.

In order to both speed up computation and to enhance motion estimation, we decided to split each frame into eight areas of concern (SSWs) around its edges (See Figure 2). The center square is not considered, as typically an object of interest is present in the central part of each frame. This is optimization for both lowering required computational power and to improve estimation of local and global motion vectors (object of interest can perform movement independent of camera's movement). If not considered, this would bring unwanted error to global motion vector. SSWs have rectangular shape proportional to the resolution of input frame. Because of the nature of jitter [25] and assuming that the input video has at least 30 frames per second, the distance from the frame's edge can be 5 % of resolution or less even for high focal lengths (e.g. $F = 600\,mm$). The size of matching sub-windows must be small enough to bring significant savings in required power and big enough to have sufficient amount of details for searching in SSWs of the previous frame. Therefore, size of the MSWs is 50 % of the SSWs size. The size of single SSW is 20 % of the frame resolution. This together with 5 % distance from the edges leaves a total of 40 % of unused space (each gap between two SSWs is 15 % of resolution big). However, the real dimensions can be adjusted based on the estimated maximal variation in each axis that should be properly stabilized (e.g. if movement is expected in only one of axes, the SSWs can be adjusted accordingly). This brings trade-off between speed and accuracy of stabilization.

Figure 3. Detail of top left area of interest from Figure 2.

However, in order to speed up GPU computation (to overcome limitations discussed in Section 2), a coalesced access to GPU's memory must be ensured. Therefore, matching sub-windows's dimensions must be rounded to the multiple of 64. This also applies to searching sub-windows: their dimensions must be updated to the double of those of matching sub-windows.

3.2 Areas of concern processing

After the previous step, eight areas of concern exist. However, adjustments must be made in order to further lower the computation requirements. This is achieved by converting each area of full-bit frame into binary image by local binary pattern (LBP) in order to enable template matching by simple XOR operation. However, conversion to binary image itself is tricky – a high level of detail must be preserved after binarization step (traditional methods tend to convert similar colors into the same binary value and therefore omit some slight contrast changes which can be considered as edges). For this, LBP binarization proposed by [17], which solves this issue, is used: each pixel of input image is compared against P equally spaced reference pixels (points) forming a circle of a radius R. Output value for each pixel of the output image is then computed:

$$B_{(P,R)(i,j)} = \begin{cases} 1 & if \sum_{p=0}^{P-1} sign(I(p) - I(i,j)) \geq \lfloor P/2 \rfloor \\ 0 & otherwise \end{cases} \quad (1)$$

and

$$sign(x) = \begin{cases} 1 & x \geq 0 \\ 0 & otherwise \end{cases} \quad (2)$$

where P is the count of reference points, R is their radius, (i, j) is the coordinate of currently processed pixel, p is the coordinate of current reference point, I is the function returning image's intensity value for given coordinate and $\lfloor x \rfloor$ denotes the largest integer not greater than x.

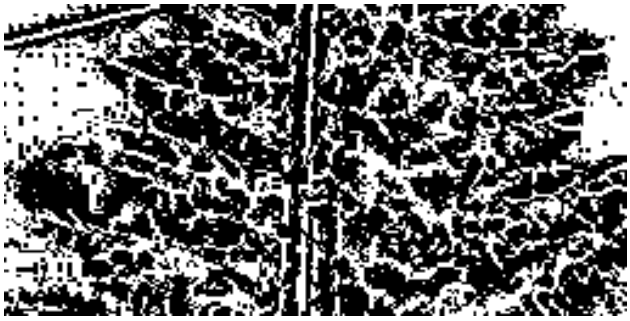


Figure 4. Searching sub-window area from Figure 3. LBP Binarization [17] successfully preserves edges of blurred image.

This approach reduces the maximum number of comparisons and additions to obtain pixel value to P. The performance of this method can be seen in Figure 4.

3.3 Local motion estimation

After the preprocessing step, local motion estimation can take place. Firstly, matching sub-windows must be extracted from the binarized areas. After that, comparison of all MSWs of current frame with the corresponding SSWs of previous frame is performed by computing number of non-matching point criteria for each possible displacement [17]:

$$NNMP(dx, dy) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \{B^t(i, j) \oplus B^{t-1}(i+dx, j+dy)\} \quad (3)$$

and

$$-s \leq (dx, dy) \leq s \quad (4)$$

where (dx, dy) is the candidate displacement of the matching sub-window in the searching sub-window, N is the MSW's dimension ($N \times N$), B^t is MSW of current frame, $B^{(t-1)}$ denotes SSW of previous frame, $\oplus$ represents Boolean operation XOR and s is half of the difference of the matching sub-windows and searching sub-windows dimensions.

This results into eight matrices of NNMP values, where each value's index denotes (dx, dy). From each list, the eight lowest values are taken and their coordinates become the local motion vectors. This is an improvement suggested by [1] in order to enable stabilization of frames without clear edges (e.g. desert, sea, snow). This gives in total of 64 LMVs vectors.

3.4 Global motion estimation and filtering

Global motion vector (GMV) is computed for each frame as a median of all 64 LMVs:

$$GMV_d = median\{LMV_i(d))\}, i = 1, 2, ..., 64 \quad (5)$$

where d is the direction vector for x and y axes ($d \in (x, y)$)

This filtration effectively removes LMVs, into which an error was introduced by object of interest movement extended into the searching sub-windows.

However, in order to preserve intended movement of camera, another filtering is needed: Typical shaky video consist of both intended motion and unwanted motion (shaking). This two motions have different properties: Shaking typically consist of fast, high frequency random changes of position (e.g. in case of hand-held camera, typical frequency is up to 20 Hz [25]). This is filtered out by four dimensional Kalman filter - current frame stabilized location (x and y) is used as a measurement for Kalman filter which yields new estimated position. Then, current frame location is compensated by difference between estimated and real position. This process achieves effective smoothing of camera path, where intended camera motion is preserved and shakiness discarded.

3.5 Motion compensation

The last step in digital image stabilization is the movement of image frames into final form of stabilized output. Because one of the requirements is possibility of real-time usage, a standard form of DIS is chosen. This consists of placing of current frame into black background on corresponding location. As a consequence, stabilized video contains disturbing black edges. In order do prevent this disturbances, a reduction of output resolution can be performed using windowing: a window with resolution lower by MSW size than that of the input video is placed into the center of each stabilized frame and non-overlapping parts are discarded.

Figure 5. Comparison of compensated frame (left) and windowed frame (right).

4. Experiments

The proposed method was tested using hardware listed in Table 1. As can be seen, used hardware is fairy old and with mediocre performance.

Table 1. Used hardware

Component	Type	Speed
CPU	Intel Core 2 Quad 6600	3.22 GHz
RAM	6 GB	1066 MHz
GPU	NVIDIA GTX 560Ti-448	930 MHz
	1.28 GB	2.20 GHz

For testing of proposed algorithm, four video sequences were created. An overview of their common measurable properties is in Table 2. Each sequence incorporates different situation and movement:

"walking" This video sequence was created by hand-held compact camera during walking through hall with bad luminance conditions. Therefore, it contains blurry frames and changing parallaxes.

"car-ride" This video sequence was recorded from camera mounted inside car. Shooting was performed during sunny weather. However, video contains focus changes between infinity and interior and rapid shaking caused by car going through potholes.

Table 2. Testing video-sequences and their properties

Video name	Resolution	FPS
walking	1280×720	30
car-ride	1280×720	30
object-tracking	1920×1080	60
pan&zoom	1920×1080	60

"object-tracking" This video sequence is shot with hand-held camera with $F = 500$ mm. It tracks movement of distant object during good light conditions.

"pan&zoom" This video sequence is also shot with hand-held camera. However, this time with the "panorama" effect (smooth position changes) and big zoom changes.

Each video sequence was stabilized using proposed method. A total of 5 different sizes of matching sub-windows was used in order to check its suitability for real-time processing. In this case, real-time processing refers to the ability to stabilize video sequence at least at the speed of 24 FPS (frames per second), which is the international standard speed used in many video cameras. The measured results are in Table 3. As can be seen, used GPU can process in real-time MSWs with resolution of 96×96 px, while there is still time left for further processing. This windows sizes enables processing of HD video with no problems and processing of FullHD video with lower amount of jitter. With age and performance of the used hardware, this can be considered as a great result. Also, it can be assumed, that use of newer, more powerful hardware will enable real-time use even for bigger MSW sizes.

Further, the proposed method performance was compared with CUDA video stabilization method implemented in OpenCV libraries [26]. The processing speed achieved 29.9 FPS (33.5 ms) for video resolution of 1280×720 px and 22.1 FPS (45.2 ms) for video resolution of 1920×1080 px, which is slower than the proposed method. OpenCV's method enables real-time stabilization of HD video with some space left for further processing. However, for FullHD video, real-time stabilization itself is not possible. The lower processing speed is due to the fact, that, OpenCV's method compensates also inter-frame rotation and uses image inpaiting to remove black edges.

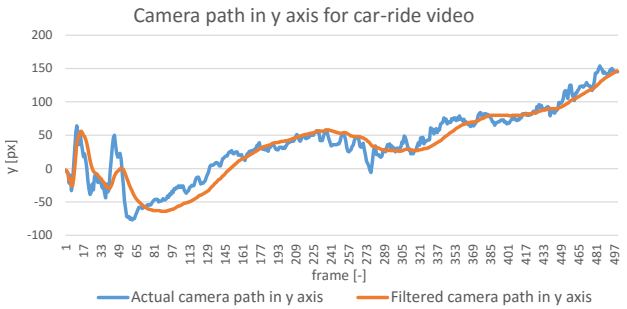


Figure 6. Motion filtration results for the *"car-ride"* video sequence: actual and filtered path of the camera motion in y axis. The filtered motion is smooth and sensitivity to the sudden short position changes declines with time.

In theory, processing speed declines quadratically with rising dimensions of matching window (this is due to the correlation complexity of $O(n^2)$). However, the testing shows, that real penalty of dimensions doubling is slightly smaller. This is due to the fact, that correlation with smaller search window resolutions is not demanding enough to fully utilize GPU.

The quality of jitter filtering and intended camera motion preservation was firstly compared with results of method available in OpenCV [26] (see table 4). For this reason, the inter-frame transformation fidelity metrics (ITF) [27] was implemented:

$$ITF = \frac{1}{N_f - 1} \sum_{k=1}^{N_f - 1} PSNR(k) \qquad (6)$$

Table 3. Relationship between matching sub-window (MSW) size and frame processing speed

MSW Resolution	FPS	Processing speed [ms]
64×64	188.2	5.3
128×64	80.0	12.5
96×96	31.7	31.5
128×128	23.0	43.5
192×96	19.0	52.6

Table 4. Stabilization quality comparison of proposed method with method implemented in OpenCV library [26].

Video name	Original ITF [dB]	Proposed ITF [dB]	Method in [26] ITF [dB]
walking	21.0	22.3	21.5
car-ride	27.7	28.9	28.0
object-tracking	21.1	21.5	22.2
pan&zoom	20.6	25.3	24.2

where N_f is the number of video frames and

$$PSNR(k) = 10log_{10}\frac{Ip_{max}}{MSE(k)} \tag{7}$$

is the peak signal-to-noise ratio between two consecutive frames, where

$$MSE(k) = \frac{1}{M \times N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\|F_k(i,j) - F_{k-1}(i,j)\|^2 \tag{8}$$

is the mean square error between monochromatic images with dimensions of $M \times N$, Ip_{max} is the maximum possible pixel intensity in the frame and F_k is the k-th frame from sequence.

As can be seen, the proposed method slightly outperforms method from [26] in three video sequences (except the pan&zoom sequence). However, subjective visual comparison with method implemented in [26] has shown, that despite the nearly the same ITF values, proposed method better compensates high frequency jitter (see [28]). Further, the difference between stabilized and shaky video sequences is subjectively much greater than is show by ITF metrics. Therefore, it can be concluded, that this metrics is not suitable for video sequences containing changes of camera position and/or tracked object position.

Next, video stabilization quality was further evaluated both visually and using discrete Fourier transform to analyze present frequencies. The visual comparison of real and filtered motion can be seen in Figures 6 and 7. It is obvious, that in case of low jitter in the *"car-ride"* video sequence, the filtering is excellent. However, in *"walking"* video sequence, filtered motion fails to straighten the camera path and as a consequence, it is dangling. Therefore, in order to address this issue, further research is required to enhance used Kalman filtering.

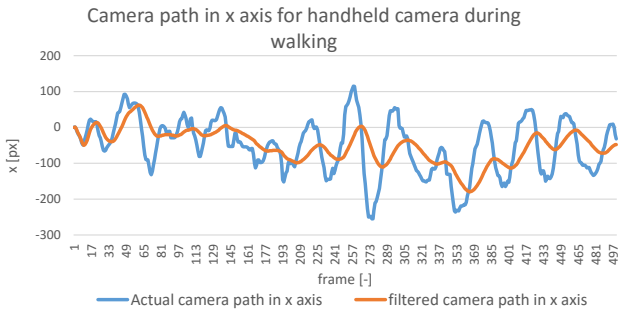


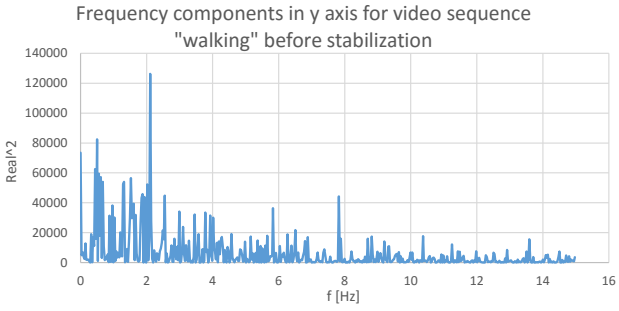
Figure 7. Motion filtration results for the *"walking"* video sequence: actual and filtered path of the camera motion in x axis. The filtered motion is smooth, but Kalman filtering fails to straighten motion caused by walking.

The analysis of frequency components for y axis of walking video sequence is shown in Figure 8. As can be seen, unfiltered motion contains a high amount of frequency changes around 1 Hz and 2 Hz. Beside those peaks, a lot of other, higher frequencies is present. However, the filtration successfully removes jitter and only low frequencies representing intended camera motion are preserved. Filtrated motion contains high amount of no motion, few smooth motions up to 1 Hz and very low amount of other frequencies. Therefore, proposed video stabilization method can be considered as an excellent.

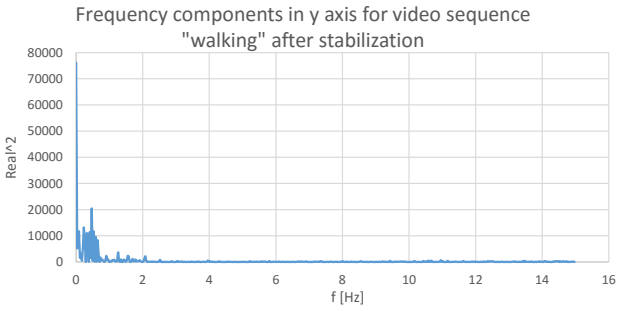
5. Conclusions

In this paper, a new method for real-time video stabilization empowering the power of GPGPU was proposed. In order to achieve real-time stabilization of HD video sequences, eight areas of interest distributed regularly around the frame edges were selected. They were further processed using fast LBP image binarization technique. This enabled fast correspondence search between two consecutive frames using simple XOR during correlation. The selection of optimal stabilization path was achieved using 64 best responses of correlation (8 responses for each area of interest), from which median value was selected for each axis. The desired shift to compensate displacement between following video frames was corrected using Kalman filtering. This enabled to preserve intended camera motion. Algorithm itself was implemented using the CUDA API.

For the testing purposes, four HD video sequences



(a)



(b)

Figure 8. Frequency analysis of the y axis path for *"walking"* video sequence: before motion filtration (a) and after motion filtration (b). Stabilization successfully removed unwanted motion of higher frequencies.

were created. They were processed using different sizes of search windows, which is the trade-off between the speed and accuracy of stabilization (low search window dimensions can cause improper stabilization in case of sudden and distant pose change). Test results on the used hardware shown, that real-time processing (24 FPS) is possible for search window sizes up to 192×192 *px* (the dimensions of matching template are half of search window dimensions; see Table 3 for detailed results). This enables successful real-time stabilization of both HD and FullHD video. Based on the used hardware (see Table 1) and its performance, it can be assumed that use of more current (and more powerful) hardware would enable real-time processing even for bigger search windows sizes. The quality of stabilization itself was evaluated using ITF metrics [27], when inter-frame difference is measured. Also, proposed method was compared with method implemented in [26] using the same hardware. This method was greatly outperformed by proposed method in terms of speed and slightly in terms of achieved ITF. Next, visual comparison of camera path before and after Kalman filtering was performed (see Figures 6 and 7). The results can be evaluated as good, because corrected camera path had smooth proceedings. However, further improvements can be achieved, as Kalman filtering fails to quickly respond to the sudden intended changes of camera pose and to straighten dangling motion of low frequency. The third evaluation method was done by frequency analysis of the camera path before and after stabilization (see Figure 8). Their comparison shows, that prior to the stabilization, frequency peaks were present over the whole spectrum with occasional peaks at certain frequencies. However, they were successfully removed by Kalman filtering and only desired motion with frequencies lower than 2 Hz was preserved.

These tests has shown, that use of GPGPU brings great performance gain and enables real-time video stabilization even on a fairly average hardware. Also, despite the simplicity of the method, the quality of video stabilization is excellent.

However, in order to use proposed method in real-life applications, further research is required. This concerns mainly filtering of unwanted motion, where Kalman filter is not sufficient and some kind of hybrid technique is required. Also, in some applications, it might by suitable to incorporate rotation angle compensation.

References

[1] M. Drahansky, F. Orsag, and P. Hanacek. Accelerometer based digital video stabilization for general security surveillance systems. *International Journal of Security and Its Applications*, 1(1):10, 2010.

[2] S. Mittal and J. S. Vetter. A survey of methods for analyzing and improving gpu energy efficiency. *ACM Comput. Surv. 47*, 2(19):23, July 2014.

[3] E. Monteiro, B. Vizzotto, C. Diniz, B. Zatt, and S. Bampi. Applying cuda architecture to accelerate full search block matching algorithm for high performance motion estimation in video encoding. *IEEE International Symposium on Computer Architecture and High Performance*, pages 128–135, 2011.

[4] S. Kumar, H. Azartash, M. Biswas, and T. Nguyen. Real-time affine global motion estimation using phase correlation and its application for digital image stabilization. *IEEE Transactions on Image Processing*, 20(12):3406–3418, December 2011.

[5] S. W. Kim, S. Yin, K. Yun, and J. Y. Choi. Spatio-temporal weighting in local patches for direct estimation of camera motion in video stabilization. *Computer Vision and Image Understanding*, 118:71–83, January 2014.

[6] M. Grundmann, V. Kwatra, and I. Essa. Auto-directed video stabilization with robust l1 optimal camera paths. *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 225–232, 2011.

[7] M. Okade and P. K. Biswas. Video stabilization using maximally stable extremal region features. *Multimedia Tools and Applications*, 68(3):947–968, February 2014.

[8] F. Liu, M. Gleicher, H.-L. Jin, and A. Agarwala. Content preserving warps for 3d video stabilization. *ACM Transactions on Graphics*, 28(3):1–9, 2009.

[9] M. J. Tanakian, M. Rezaei, and F. Mohanna. Real-time video stabilization by adaptive fuzzy filtering. *International Conference on Computer and Knowledge Engineering*, pages 126–131, 2011.

[10] Z. H. Zhou, H. L. Jin, and Y. Ma. Plane-based content preserving warps for video stabilization. *IEEE international conference on Computer Vision and Pattern Recognition*, pages 2299–2306, 2013.

[11] K. Ohyu, S. Jeongho, and P. Joonki. Video stabilization using kalman filter and phase correlation matching. *Image Analysis and Recognition*, pages 141–148, 2005.

[12] A. Litvin, J. Konrad, and V. C. Karl. Probabilistic video stabilization using kalman filtering and mosaicking. *SPIE Symposium on Image and Video Communications and Processing*, pages 663–674, 2003.

[13] Y. G. Ryu and M. J. Chung. Robust online digital image stabilization based on point-feature trajectory without accumulative global motion estimation. *IEEE SIGNAL PROCESSING LETTERS*, 19(4):223–226, 2012.

[14] C. T. Wang, J. H. Kim, K. Y. Byun, J. Q. Ni, and S. J. Ko. Robust digital image stabilization using the kalman filter. *IEEE Transactions on Consumer Electronics*, 55(1):6–14, 2009.

[15] C. H. Song, H. Hai, W. Jing, and H. B. Zhu. Robust video stabilization based on particle filtering with weighted feature points. *IEEE Transactions on Consumer Electronics*, 58(2):570–577, 2012.

[16] J. Yang, D. Schonfeld, and M. Mohamed. Robust video stabilization based on particle filter tracking of projected camera motion. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):945–954, 2009.

[17] B. Kir, M. Kurt, and O. Urhan. Local binary pattern based fast digital image stabilization. *Signal Processing Letters*, 3(22):341–345, 2015.

[18] NVIDIA. Cuda c best practices guide, September 2015. http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.htm.

[19] AMD. Amd accelerated parallel processing - opencl programming guide, November 2013. http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf.

[20] NVIDIA. Cuda 4.0, 2011. http://developer.download.nvidia.com/compute/cuda/4_0/CUDA_Toolkit_4.0_Overview.pdf.

[21] NVIDIA. Cuda 5.0, 2012. http://on-demand.gputechconf.com/gtc/2012/presentations/SS104-CUDA-5-What's-New.pdf.

[22] NVIDIA. Cuda 6.0, April 2014. http://devblogs.nvidia.com/parallelforall/powerful-new-features-cuda-6/.

[23] NVIDIA. Cuda 7.0, January 2015. http://devblogs.nvidia.com/parallelforall/cuda-7-feature-overview/.

[24] NVIDIA. Cuda tools and ecosystem. https://developer.nvidia.com/cuda-tools-ecosystem.

[25] F. L. Rosa, M. C. Virzi, F. Bonaccorso, and M. Branciforte. Optical image stabilization.

[26] OpenCV. Video stabilization, December 2015. http://docs.opencv.org/trunk/d5/d50/group__videostab.html#gsc.tab=0.

[27] J. Xu, H. W. Chang, S. Yang, and M. Wang. Fast feature-based video stabilization without accumulative global motion estimation. *IEEE Transactions on Consumer Electronics*, 58(3):993–999, 2012.

[28] Pacura, D. Car-ride cuda video stabilization comparison with opencv implementation, March 2016. https://youtu.be/Z5FDnbTbfSE.