SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 14, No. 1, February 2017, 67-84

UDC: 004.25 **DOI: 10.2298/SJEE1701067P**

A Method for Real-Time Memory Efficient Implementation of Blob Detection in Large Images

Vladimir L. Petrović[1], Jelena S. Popović-Božović[1]

Abstract: In this paper we propose a method for real-time blob detection in large images with low memory cost. The method is suitable for implementation on the specialized parallel hardware such as multi-core platforms, FPGA and ASIC. It uses parallelism to speed-up the blob detection. The input image is divided into blocks of equal sizes to which the maximally stable extremal regions (MSER) blob detector is applied in parallel. We propose the usage of multiresolution analysis for detection of large blobs which are not detected by processing the small blocks. This method can find its place in many applications such as medical imaging, text recognition, as well as video surveillance or wide area motion imagery (WAMI). We explored the possibilities of usage of detected blobs in the feature-based image alignment as well. When large images are processed, our approach is 10 to over 20 times more memory efficient than the state of the art hardware implementation of the MSER.

Keywords: Real-time blob detection, Maximally stable extremal regions, Parallelism, Multiresolution analysis, Image alignment.

1 Introduction

In computer vision, blob detection is usually defined as a detection of regions in an image that possess some distinguishing properties compared to surrounding regions or to the image background. For example, characteristic properties of those so-called blobs can be brightness or color. Blob detection is one of the basic parts of a lot of image analysis systems. Blobs are often already the objects we want to detect, e.g. some particles, cells in medical imaging, characters in text recognition etc. However, sometimes it is impossible to determine whether a detected blob is a desired object by using simple detection. In such cases, the detected regions are usually the input to the other stage of the object detection algorithm, e.g. moving car detection and tracking in the video surveillance applications. The detected blobs are also frequently used as distinctive image features for the image matching together with the SIFT [1], SURF [2], BRIEF [3] or other descriptors. Many of these applications require real-time performance which can be achieved in software only on high

[1]University of Belgrade, School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia; E-mails: petrovicv@etf.rs; jelena@etf.rs

processing power platforms. Hence, if needed in low power embedded systems, the existing blob detection algorithms need parallelization in order to achieve the required performance with low energy and memory cost.

Well known methods for blob detection are Laplacian of Gaussian, the difference of Gaussians and the determinant of Hessian approach, including their affine and hybrid versions. Other common methods for image segmentation and region detection are watershed algorithms. Typical example of the watershed algorithm is a blob detection method of maximally stable extremal regions (MSER) [4]. The MSER detection algorithm is able to detect both fine and coarse blobs, i.e. both small and large objects. It is used in applications such as cell detection in medical imaging [5], automatic 3D-reconstruction from a set of images [6], feature detection and matching [7], or in automated surveillance systems for object detection and tracking [8].

Although the MSER detection algorithm is suitable for many applications, its computational cost is high and limits its real-time performance only for low resolution images. State of the art FPGA implementation has real-time performance, but only for images with spatial resolution up to 350 × 350 pixels [9]. The recent ASIC implementation has better performance, but it is designed to work on a higher clock rate [10]. Those two implementations have similar speed performance if comparison is based on the number of operations per clock cycle.

We propose a method for blob detection which uses the MSER detector from [9], but applied to blocks of the divided input image in parallel. Parallelism provides a great speed-up of the detection algorithm. If applied to a large input image, the MSER detector from [9] would not be able to achieve real-time performance and its implementation's processing memory requirement would be extremely large. In our method, we use only small blocks of the image for calculation, hence the processing memory cost is significantly reduced. The limitation of this approach is the inability to detect blobs whose size is larger than the block size. Also, for some applications, it is unable to detect large blobs at the borders of blocks. We described that problem in our previous conference paper [11]. In this paper, we overcome it by proposing the multiresolution analysis which increases the initial memory cost, but enables detection of larger blocks. We believe that this method can be used in many applications, whether used with or without the multiresolution analysis.

In the next section, we briefly describe the MSER detection algorithm and its FPGA implementation from [9] which is used as a reference for this work. The method for parallel image processing, some possible applications and analysis of performance and memory usage are described in Section 3, which is the main contribution of the paper. In Section 4 we present other applications of this approach and use the detected MSER regions for the feature-based image

alignment. Finally, we summarize our results and give conclusions and proposals for further work in Section 5.

2 Maximally Stable Extremal Regions

2.1 Definition of the maximally stable extremal regions

In this paper we used 8-bits per pixel grayscale images. If we apply a threshold at every possible pixel level $t \in [0,\ 255]$ to the image I, we get a set of binary images as the result of the calculation

$$I_{\text{bin}}^{t} = \begin{cases} 1, & I \geq t \\ 0, & I < t \end{cases}. \tag{1}$$

In each binary image, we can see the set of connected regions that are called extremal regions. If we look at this set of binary images, the extremal regions at lower threshold are divided into multiple smaller extremal regions as the threshold increases. These extremal regions create a component tree as shown in Fig. 1. Each node of the tree represents a connected region at threshold t noted as R_j^t, where j is the number of the region. Size of the region i.e. number of pixels in the region is $\left|R_j^t\right|$. We can observe the region R_j at different threshold values by looking at one branch of the component tree.

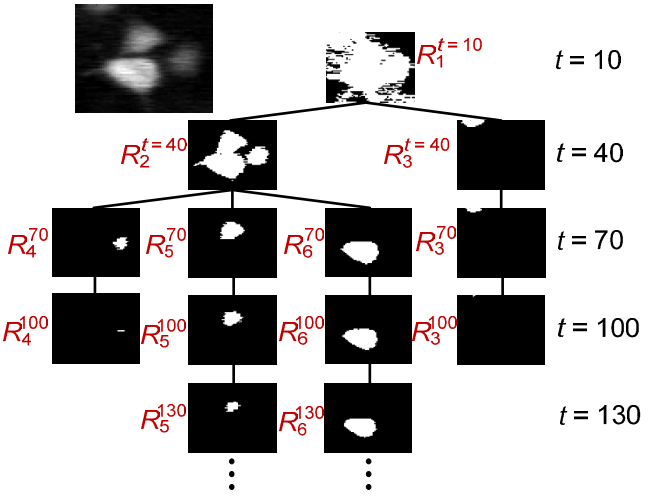


Fig. 1 – *A part of the regions tree for determining maximally stable extremal regions, for an example image in the upper-left corner. The complete regions tree contains regions for all possible thresholds.*

The region is maximally stable if the stability factor $q(t)$ defined as

$$q(t) = \frac{\left|R_j^{t-\Delta}\right| - \left|R_j^{t+\Delta}\right|}{\left|R_j^t\right|} \tag{2}$$

has a local minimum at t^*, where Δ is the parameter of the method. The authors of the MSER blob detection method define the parameter called maximal value of the stability factor $q_{\max}$ [4]. If a region has the stability factor $q(t^*)$ larger than $q_{\max}$, it should be rejected although it has a local minimum at t^*. The larger the $q_{\max}$ is, the more MSER regions are detected, but the detected regions are less stable. In these paper, we chose $q_{\max} = 0.25$. This analysis applies to detection of the bright regions on a dark background, while the analysis of the inverted input image $I_{\text{inv}} = 255 - I$ gives the dark regions on a bright background.

2.2 Implementation of the MSER algorithm

Algorithm for MSER detection can be divided into three basic stages. The first one is preprocessing, where the intensity level histogram of an image is calculated and pixels are sorted by intensity. The sorting is done by using the bin sort algorithm [12], since it is very efficient if the intensity level histogram is known before the sorting starts. The second stage is clustering at which the representation of all regions at each threshold is created. This is done by using the Union-find algorithm [12] which is used to keep track of the regions of connected pixels. The final stage is tracking the sizes of the regions and their stability factors. Local minimums of the stability factor determine the maximally stable extremal regions.

As a reference design in this paper we use the implementation of the MSER algorithm described in [9]. At the beginning of processing, the pixels are sorted. If the image has N pixels, the sorted pixels positions are written to the N-entry memory where each entry has $\log_2 N$ bits. When the sorting is finished, each pixel in the image is processed in sorted order. The algorithm uses a memory which is called the Region Map (RM). The region map has N memory locations too. Each memory location has three numbers that are used to keep track of which pixels are added to which region, which pixels belong to a single region and which pixels have already been processed. The first number is called the union-find number (U). If this number is equal to 0, it means that the pixel is not connected to any other pixel or that the pixel has not yet been processed. If $U > 0$, the pixel is a member of the same region as the pixel at position U. Finally, if $U < 0$, the pixel is the reference point of the region and $1 - U$ is the region size (number of pixels in the region). Union-find number U is a $1 + \log_2 N$ bits long word.

A single bit is added to each region map location and it is an indicator that shows whether the pixel is processed or not.

In order to speed-up determining which pixels belong to the region with the reference point at the location *p*, every region has the linked list of pixels in that region. This means that each entry in the region map has additional $\log_2 N$ -bit number which is a pointer to the next pixel in the list.

An example of adding a pixel at level $t = i - \Delta - 1$ to the region map is shown in Fig. 2. When processing the pixel, we check right, up, left and down neighboring pixels. If a neighbor belongs to an existing region ($U > 0$ or $U < 0$) we add the current pixel to that region. Otherwise, we check if the neighbor is already processed. If it is not, that means that it has lower intensity value than the current pixel and it is therefore skipped. If it is processed, a new region is made from the current processing pixel and the neighboring pixel. The example in Fig. 2 shows the most complex situation when a single pixel causes merging of two regions.

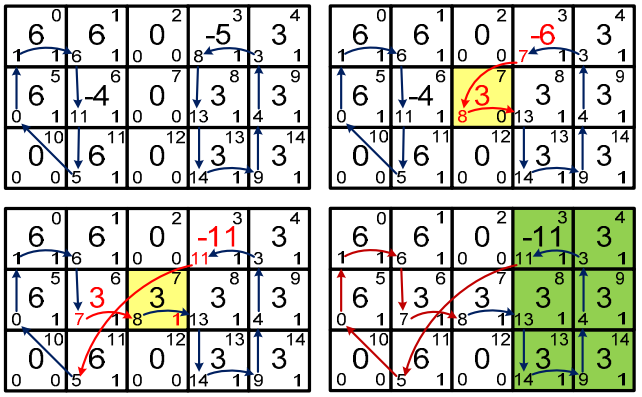


Fig. 2 – *A region map (RM) for the union-find operations. Each RM memory location represents one pixel. The large middle number in each memory location is the union-find number (U). The number in the upper-right corner is the pixel address, and the number in the lower-right corner is an indicator that shows whether the pixel is processed (1) or not (0). Number in the lower-left corner is the address of the next pixel in the linked list of connected pixels in the region. The example here (taken from [9]) shows processing of the pixel at position 7 whose intensity is* $i - \Delta - 1$. *The upper-left image shows the RM at intensity i. Initially, the pixel at position 7 is added to the region on the right due to the first neighbor check at the right side. After the neighbor check at the left side, two regions merge since the processing pixel needs to be added to both of neighboring regions. In case we need region pixels at threshold* $t = i$, *first links are bypassed, while needed pixels are shadowed ones, like it is shown in the lower-right image.*

In order to keep track of the sizes of connected regions, a hash indexed memory is used. Whenever all pixels from one intensity level have been processed, the size of all regions that grew is updated in this memory. Sizes for the region R_j are kept only for the intensity levels from $t-\Delta-1$ to $t+\Delta+1$. These intensity levels are needed for the calculation of stability factors $q(t-1)$, $q(t)$ and $q(t+1)$. If these three stability factors are known, we can check if the $q(t)$ is a local minimum. If it is a local minimum, then the region R_j^t is the maximally stable extremal region. For further details about the implementation, please refer to [9].

3 Parallelism for the Detection Speed-up and Reduced Memory Cost

In this section we propose a system for real-time detection of the MSER blobs with reduced memory cost. The system realizes the concept of parallelism which enables high-speed performance, while dividing large image into smaller blocks induces low memory cost. Input image is firstly divided into smaller squared blocks during reading from the sensor or from the memory where the image is stored. The processing is now done on these smaller blocks in parallel which induces the great speed-up of the original MSER implementation from [9]. However, the processing of the small blocks makes the missed blob detections, since the large blobs and the blobs at the edges of the blocks can be skipped. This problem can be partially solved by overlapping the blocks and by using the multiresolution analysis which increases the reliability of the method. Multiresolution analysis is performed by resizing the input image to smaller resolution images and by applying the blob detection to these resized images. This is how large objects can be detected using small block detectors. Further in this paper, multiresolution analysis is offered as optional and analyzed separately since there are applications [8] where the objects of interest are much smaller than the block size and where the multiresolution is not needed. Note that we tested the algorithm in software and have done a performance and memory cost analysis, but we leave the FPGA or ASIC implementation for future work.

3.1 System description

Block diagram of the proposed system is shown in Fig. 3. The system contains several independent processors for each scale of the multiresolution analysis. If the multiresolution analysis is used, the input image is decimated by a factor 2^{S-1}, where $S \in \{1,\ 2,\ 3,...\}$ is the scale number. The original image corresponds to the scale $S=1$. Firstly, let's consider the single scale blob detection processor, called Scale S processor, where S is the corresponding image scale.

Each processor contains M independent MSER detectors described in Sub-section 2.2. The inputs of each detector are the image blocks that can be overlapping or non-overlapping. Overlapping increases the computational load since more processing blocks are needed for the same image dimensions. Yet it reduces the number of missed detections at many applications. It is recommended that the blocks are squared and have dimensions that are powers of 2. With these dimensions, some important operations like dividing and multiplying by the block width or height are simple right or left shifts.

As the image stream is being read from the camera or some local memory, the Controller of the image read gets the pixels data for a number of lines and writes them to the M image block memories of the Scale 1 processor ($M1_1$ to $M1_M$). The MSER detectors use the data from these memories for processing while the Controller writes next lines in the second set of M memories $M2_1$ to $M2_M$. When the processing of the first set of memories is finished, the MSER detectors use the second set of memories as input. Now, the Controller again writes the new set of data to the first set of memories etc.

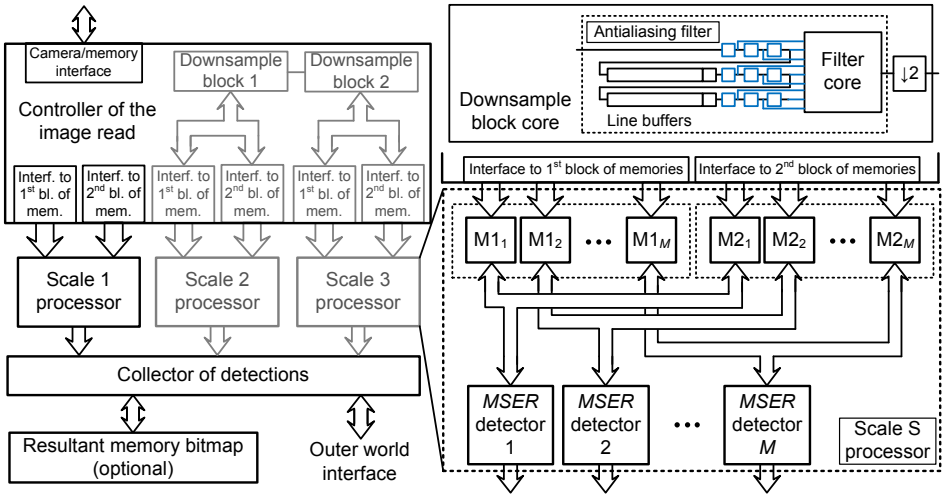


Fig. 3 – *Block diagram of the proposed system. Gray blocks are only present if the multiresolution analysis approach is used. The example shows the hardware for only 3 scales of the input image.*

When new MSER is detected, the MSER detector sends the pixel positions of the new MSER to the Collector of detections. The Collector can use this new detection for the post processing or reject it if it is invalid. Also, it can just bypass the new detected blobs via the outer world interface to the other system that uses them as input for some more complex processing.

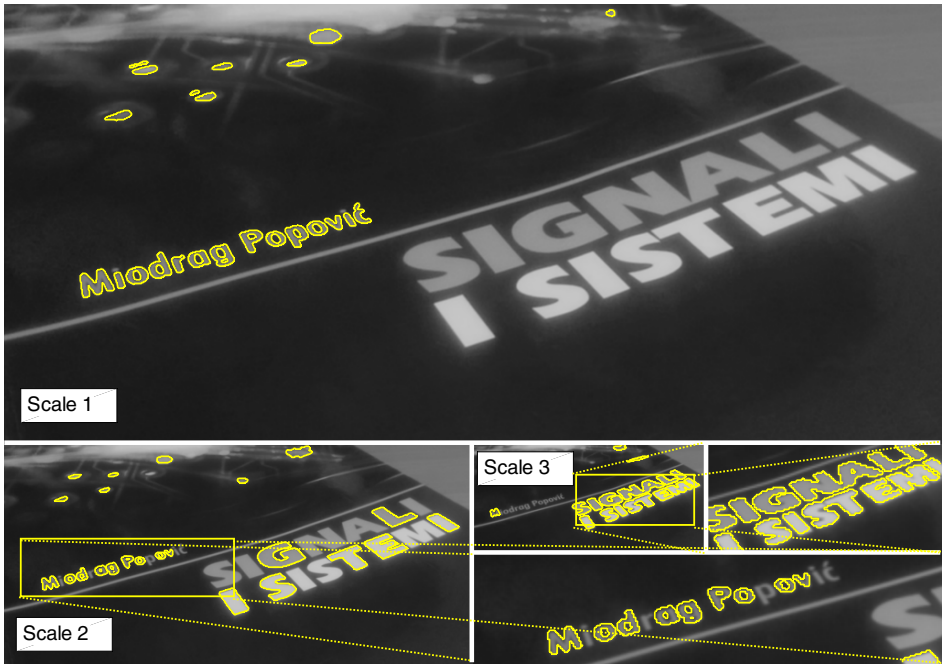


Fig. 4. – *Results of the multiresolution blob detection in the text recognition application. The example is the photo of a cover page of a textbook on which title letters are larger than author's. Small letters are detected in scales 1 and 2, while large letters are detected at scales 2 and 3. The block size is* $N_{\text{block}} = 64 \times 64$ *with the overlapping strip width of* $w_{\text{ol}} = 18$ *pixels. The parameter* Δ *is set to 3.*

While the Controller of the image read writes the pixels data to the memories in the Scale 1 processor, the Downsample block 1 takes these pixels and calculates the decimated version of the input image. Decimated image is further written to memories in the Scale 2 processor and forwarded to the Downsample block 2 for further decimation. The Downsample block 2 does the same job as the Downsample block 1, just at the lower frequency. If there is a need for even smaller image resolution, the additional Downsample blocks and Scale processors may be added.

Downsampling is done by simple decimation of the input image. In order to prevent aliasing, the input image is filtered using the 3×3 Gaussian mask with the standard deviation $\sigma = 1$. Note that for filtering, the line buffers are needed as shown in the Fig. 3. The example in Fig. 4 shows the bright-on-dark blob detection in the cover page of a textbook. The blobs of interest are the letters in author's name and textbook title. Since the image contains letters of different sizes, large letters cannot be detected at the original image scale; hence the

multiresolution analysis must be used. It is shown that at all scales from 1 to 3, almost every letter blob is detected. The parameter Δ is set to 3, the block size is set to $N_{\text{block}} = 64 \times 64$ while the overlapping strip width w_{ol} is set to 18 pixels. This approach gives good results for our example in Fig. 4, although some other approaches like bilinear or bicubic interpolation could be used.

When we use the non-overlapping image partitioning, there is a chance that a single region positioned at the block border is divided and detected as two or more neighboring regions as shown in the top row of Fig. 5. Some of these border regions could be the product of image dividing if we cut a piece of large background. These detections are false detections. This is why we sometimes should use the image partitioning with the overlapping for detection of small objects and reject all border detections. In these cases, even when using the multiresolution, the method can skip some blobs. This is a limitation that is not crucial for many applications shown in this paper.

3.2 Merging of border regions when the type of object is known

In medical imaging the MSER detection is commonly used for cell detection. Cells are usually light or dark blobs on the uniform background, hence all the MSER detections in this kind of images refer to the cells [5]. In the situations like this, we can use non-overlapping image partitioning, detect multiple region parts in multiple blocks and then merge these parts into one region. If the cell sizes do not differ too much, we can also avoid using the multiresolution analysis.

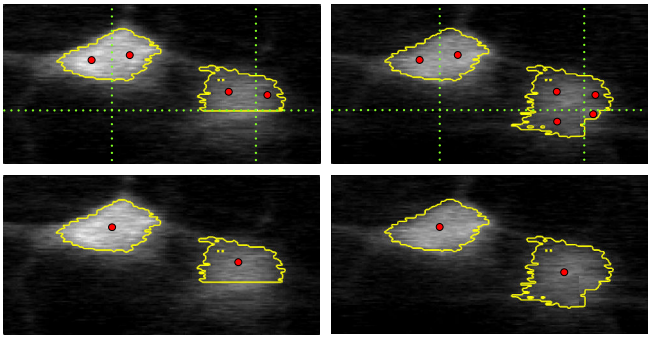


Fig. 5 – *Connecting of border detections into one region. Dots inside regions represent centroids of regions.*

In order to merge partial regions into a single region, we use a Resultant memory bitmap whose capacity is N bits, where N is the number of pixels in the input image. Each bit represents one pixel in the input image and is set to 1 if that pixel is a part of any detected MSER. During the detection in the MSER detector, we keep information whether the detected MSER is the MSER at the

border of the block and forward that information together with the region pixels to the Collector of detections. If the detected MSER is the MSER at the block border, the Collector of detections checks in the Resultant bitmap if there is a detected MSER in the neighboring block. If this is true, the current MSER is merged with the neighboring one. The neighboring region is determined by finding the shortest Euclidean distance between the current region and regions in the neighboring block. The example is shown in Fig. 5. Merging of border regions allows us to detect almost all possible blobs for applications like these.

3.3 Performance analysis

Since we have not implemented the algorithm on any target platform (FPGA, GPU, ASIC), yet only in the software, we base our performance analysis on the analysis from [9].

Based on the analysis from Sub-section 2.2 and [9], the needed memory cost for image storing and implementation of the MSER detection in an N-pixel image is

$$\begin{aligned} M_{\text{MSER}} &= M_{\text{image}} + M_{\text{sort}} + M_{\text{region\_map}} + M_{\text{result\_bitmap}} \\ &\approx 8N + N\log_2 N + N\left(1+1+\log_2 N + \log_2 N\right) + N \\ &= \left(11 + 3\log_2 N\right) N \text{ bits.} \end{aligned} \tag{3}$$

According to that, the needed memory cost for one block processing is

$$M_{\text{MSER\_block}} \approx \left(10 + 3\log_2 N_{\text{block}}\right) N_{\text{block}} \text{ bits}, \tag{4}$$

where N_{block} is the number of pixels in one block. Note that now we have number 10 inside the brackets, since in [9] N bits are needed for the resultant memory. We firstly analyze the needed processing memory. The number of bits for the resultant memory will be added in the end.

Let's consider a case when a single scale is used, i.e. the MSER detection is done only in the original input image. If the image is squared, which we will consider for simplicity, and if there is no overlapping, then the number of the processing blocks is $PB_{num} = \left\lfloor \sqrt{N} \Big/ \sqrt{N_{\text{block}}} \right\rfloor + 1$. If we use overlapping, the number of processing blocks is $PB_{num} = \left\lfloor \sqrt{N} \Big/ \left(\sqrt{N_{\text{block}}} - w_{\text{ol}}\right) \right\rfloor + 1$, where w_{ol} is the width of overlapping strip. Therefore, the total memory cost is

$$M_{\text{MSER,tot}} \approx N + N_{\text{block}}\left(10 + 3\log_2 N_{\text{block}}\right)\left(\left\lfloor \frac{\sqrt{N}}{\sqrt{N_{\text{block}}}} \right\rfloor + 1\right). \tag{5}$$

If we use the multiresolution analysis, we need to calculate the additional memory cost in other scale processors. Since the size of the block is the same

for the whole system, the memory cost for processing of one block remains the same. Since the line at scale S is 2^{S-1} times shorter than at the original scale, the number of the processing blocks for the scale processor at scale S is

$$PB_{num,S} = \left\lfloor \sqrt{N} \Big/ \left(2^{S-1} \sqrt{N_{\text{block}}} \right) \right\rfloor + 1 . \tag{6}$$

The resultant memory at scale S needs $2^{2(S-1)}$ times less bits for storing the result bitmap. This gives us the total memory cost for the multiresolution processing up to the scale $S = S_{max}$

$$M_{\text{MSER,tot,multi}} = \sum_{S=1}^{S_{\max}} \left[\frac{N}{2^{S-1}} + N_{\text{block}} \left(10 + 3\log_2 N_{\text{block}} \right) \left(\left\lfloor \frac{\sqrt{N}}{2^{S-1} \sqrt{N_{\text{block}}}} \right\rfloor + 1 \right) \right]. \tag{7}$$

Processing memory cost for different levels of multiresolution analysis depending on the squared image resolution is shown in Fig. 6. The maximum image size in this example is 4 megapixels. For image this large, the maximum number of scales, where the last scale image is smaller than or equal to the block size, is $S_{\max} = 7$.

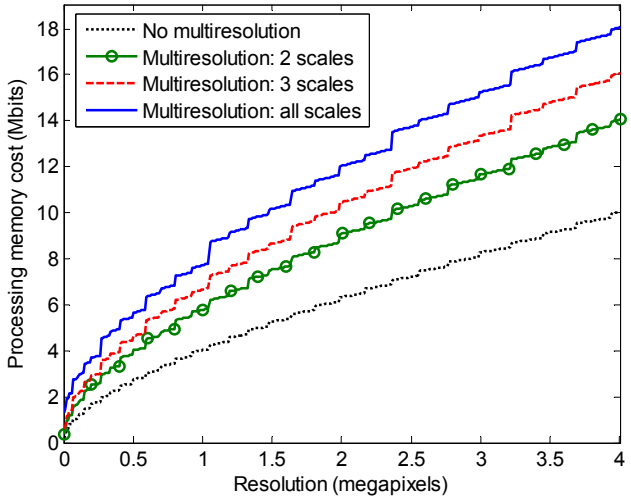


Fig. 6 – *Processing memory cost using the multiresolution approach depending on the resolution of an input image for different number of calculated scales* $N_{\text{block}} = 64 \times 64$.

Execution time of the MSER detection in [9] is approximated to $t_{\text{exe}} \approx 10NT_{\text{CLK}}$, where T_{CLK} is the clock period, but the algorithm only detects either bright or dark regions. In order to detect both the bright and the dark regions, the needed time is $t_{\text{exe}} \approx 20NT_{\text{CLK}}$. In our approach, the input image is

divided into horizontal stripes which are further divided into equal sized blocks. Therefore, the total execution time is the time needed for processing of one stripe multiplied by the number of stripes. The blocks in one stripe are processed in parallel, hence the time needed for processing of one stripe is equal to the processing time of one block $t_{\text{exe,stripe}} = t_{\text{exe, block}} \approx 20 N_{\text{block}} T_{\text{CLK}}$. Since the image in our example is squared, the number of stripes is equal to PB_{num}, hence the total execution time is

$$t_{\text{exe}} = 20 N_{\text{block}} T_{\text{CLK}} PB_{num} \approx 20 N_{\text{block}} T_{\text{CLK}} \left(\left\lfloor \sqrt{N} \Big/ \sqrt{N_{\text{block}}} \right\rfloor + 1 \right). \tag{8}$$

The Scale 2 processor and other scales processors work with the decimated image. The execution time for the lower resolutions is smaller than the execution time needed for processing of the image at Scale 1. Since all these processors work in parallel, the total execution time of the multiresolution processing is determined by the execution time at scale 1.

We summarize our estimations in **Table 1** and compare them to the state of the art FPGA and ASIC implementations from [9] and [10]. The results for the squared image of 1536×1536 pixels are calculated in the case when there is no overlapping. The execution time and memory cost are greater when blocks are overlapping. However, there is still significantly large reduction of both performance parameters comparing to the referenced implementations.

Table 1

Performance comparison with the state of the art MSER detector hardware implementations based on the squared image example.

Performance Metric	FPGA [9]	ASIC [10] (expected)	This work (expected)	This work - multiresoluton (expected)
MSER regions	Either bright or dark	Bright and dark	Bright and dark	Bright and dark
All MSER regions?	Yes	Yes	No	No
Processing memory cost (bits, approx.)	$N \times (11 + 3\log_2 N)$	$N \times (9 + 2\log_2 N)$	$N + M_{\text{block}} PB_{num,1}$	$\sum_{S=1}^{S_{\max}} \frac{N}{2^{S-1}} + M_{\text{block}} PB_{num,S}$
			$M_{\text{block}} = N_{\text{block}} \left(10 + 3\log_2 N_{\text{block}}\right)$ $PB_{num,S} = \left\lfloor \sqrt{N} \Big/ \left(2^{S-1} \sqrt{N_{\text{block}}} - w_{\text{ol}}\right) \right\rfloor + 1$	
Execution time	$\approx 10 N T_{\text{CLK}}$	$\approx 10 N T_{\text{CLK}}$	$\approx 20 N_{\text{block}} T_{\text{CLK}} PB_{num,1}$	
For: N=1536×1536 and Nblock = 64×64 ⇒ PBnum,1 = 24, Smax = 6, no overlapping				
Memory cost:	176 Mbits	121 Mbits	7.07 Mbits	13.92 Mbits
Frame rate: $f_{\text{CLK}} = 50$ MHz	2.12 fps	2.12 fps	25 fps	25 fps

Additional comparison with the implementations from [9] and [10] are shown in the Figs. 7 and 8. Fig. 7 shows extremely high memory cost efficiency of our approach in comparison with the referenced MSER detection implementations. Fig. 8 shows comparison of the frame rate for different resolutions of the input image. For the block size $N_{\text{block}} = 64 \times 64$, we can achieve real-time performance for the maximal image resolution $N_{\max} = 1536 \times 1536$, when detecting both the bright and the dark regions. Note that if we detect only bright or only dark regions, we can achieve much higher frame rate. Likewise, the memory cost for the maximal image resolution is reduced about 25 times compared to [9] and about 17 times compared to [10], if processed at only one scale.

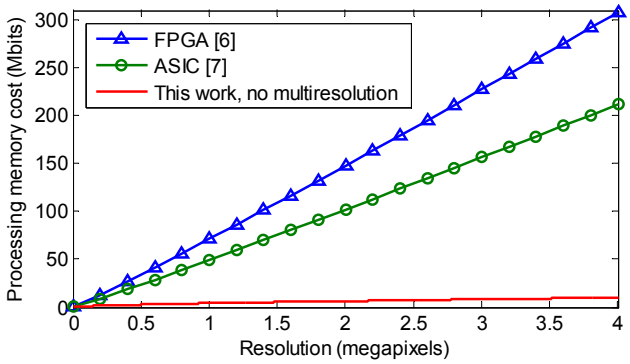


Fig. 7 – *Processing memory cost depending on resolution of an input image for reference designs from [9] and [10] and for our approach where* $N_{\text{block}} = 64 \times 64$.

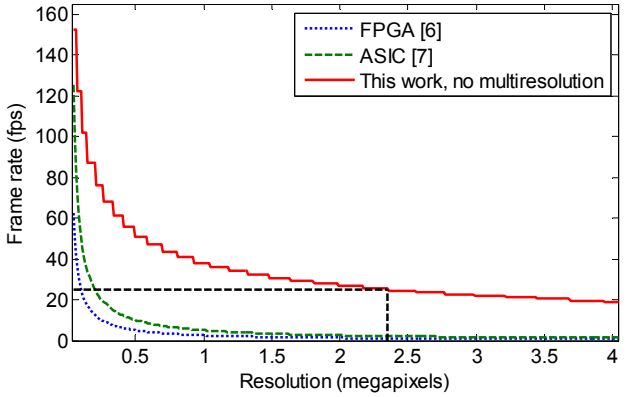


Fig. 8 – *Approximated frame rate depending on the resolution of the input image for the reference designs from [9] and [10] and for our approach. The execution time is approximated for detection of both the bright and the dark regions.*

4 Applications in Video Surveillance and Image Alignment

Besides numerous applications of the MSER detector which we mentioned before in the introduction and in Sub-sections 3.1 and 3.2, we analyzed the usage of the detected MSER blobs in the feature based image alignment. The image alignment is an important step in many image processing and computer vision applications. We found our MSER detector particularly suitable for applications such as video surveillance and wide area motion imagery (WAMI) [8]. In wide area motion imagery, the image covers a large area. In these cases objects of interest are usually small objects. The MSER blob detection is used in [8] for objects tracking when a camera is placed on the flying drone. The camera is far from the objects and hence objects are small which makes our method suitable for fast detection. In order to track moving objects, there is a need for image alignment since the drone is slightly moving. The image alignment is done by using the non-moving detected blobs as features for feature-based image alignment. This can be very convenient, since we can spare time for feature detection in the feature-based alignment, by taking already detected blobs for the image features. We were not able to get usable WAMI data, so we tested the image alignment applications using the multiple images taken on the ground by the DSLR camera in burst mode.

Fig. 9 – *Detected MSER features in the example image.*

Feature-based image alignment [13] is done in several stages. First stages are feature detection, feature description and feature matching. Afterwards, the geometric relationship between the two images is found based on the matched features. Finally, the alignment of the second image to the first one is done by its geometric transformation using the found geometric relationship. Feature detection is already done by detecting blobs using the proposed design. The example image with the detected blobs is shown in Fig. 9. In this example we used the detection without multiresolution, hence only small regions are detected.

To demonstrate that our features can be used for this application, we apply the SURF descriptor [2] to each detected region in both images. After the extraction of SURF features, the matching is done and the pairs of matched features in the first and the second image are formed. Matched MSER/SURF features in two images are shown in Fig. 10.

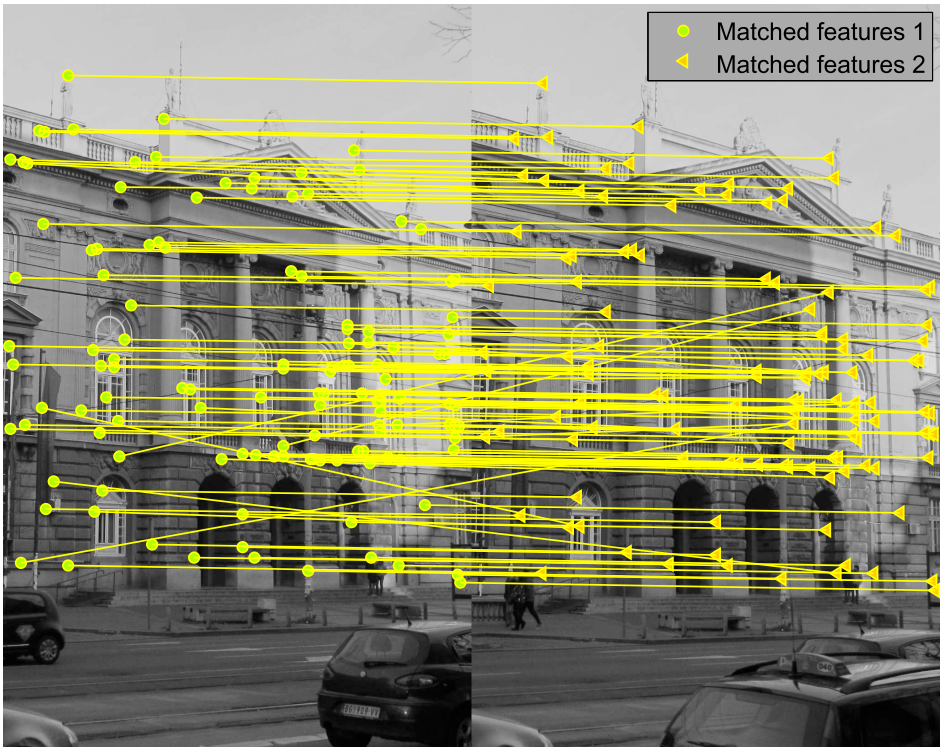


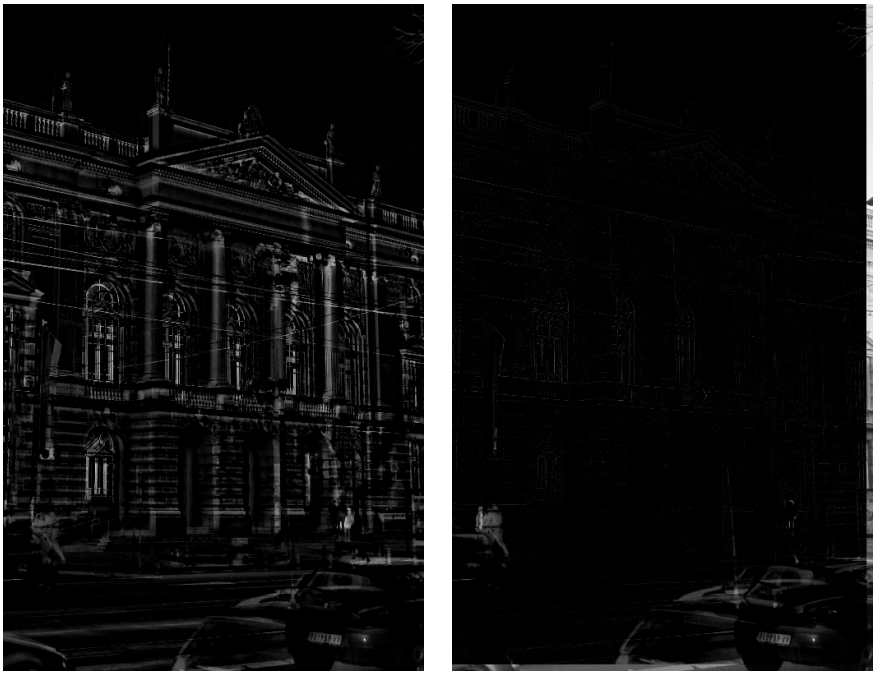
Fig. 10 – *Matched MSER/SURF features of original and shifted image used for the feature-based image alignment. Note that there are some false matches, but that most of them are correct.*

After the feature matching is done, the geometric relationship between the two images is estimated by using the M-estimator SAmple Consensus (MSAC) algorithm described in [14]. The second image is then transformed using the estimated geometric transformation.

In order to determine the quality of alignment, for quality metric, we choose the mean squared error (MSE) of all pixels in the aligned second image as compared to the pixels in the first image. The MSE is calculated as

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\left(I_{2,\text{aligned}}(p) - I_1(p)\right)^2 . \tag{9}$$

For the example shown in Figs. 9 and 10, the initial mean squared error of non-aligned images is equal to $MSE_{\text{original}} = 69.8$. After the feature-based alignment is done, with the MSER detection from this paper, block size $N_{\text{block}} = 64 \times 64$, overlapping strips of 8 pixels wide and by using only first scale, we get the mean squared error $MSE_{\text{alligned}} = 15.7$.



(a) (b)

Fig. 11 – (a) *The difference between non-aligned images* $I_1 - I_2$; (b) *The difference between aligned images* $I_1 - I_{2,\text{aligned}}$ *with the MSER detection from this paper.*

The differences between non-aligned images and between aligned images are shown in Figs. 11.a and 11.b. Note that there is a significant difference between the two aligned images in the bottoms which increases the MSE. This is the product of moving objects in the image. These moving objects have no influence to the alignment result. We compared the MSE when detection is done by using our approach and when detection is done by the conventional MSER detection algorithm. We could not see any differences in alignment results except those that are caused by the statistical properties of the MSAC algorithm.

5 Conclusion

In this paper we described the method for the memory efficient blob detection based on the MSER algorithm which can work in real-time for large images. The method significantly outperforms the state of the art MSER hardware detection realizations in terms of the needed processing memory and the frame rate, but it induces smaller number of the detected regions. Dividing the image into the smaller blocks, even when overlapping, makes the method unable to detect large blobs. However, we gave some examples in medical imaging, wide area motion imagery and feature-based image alignment which show that the method can still give good results, but much faster and with reduced memory cost compared to other realizations. This was the main contribution of our previous conference paper [11]. Additionally, we explored possibilities for multiresolution analysis of the image and proposed the system that can detect large objects as well. A good example for this application is the real-time letters detection when the text contains letters of different sizes.

We believe that, with proper setting of parameters (the number of scales in the multiresolution analysis S, the size of a block N_{block}, and overlapping strip width w_{ol} at first), this approach can be used in many other applications too. The algorithm provides the space for compromise between accuracy and the number of detected regions, on one side and the memory cost and the execution speed, on the other side.

In the future work we plan to implement our parallel algorithm on an FPGA platform and explore more possibilities and new applications of this approach.

6 Acknowledgement

We would like to thank Dragomir El Mezeni from the University of Belgrade - School of Electrical Engineering, and Prof. Dejan Marković and Dejan Rozgić from the University of California, Los Angeles for useful suggestions and comments.

7 References

[1] D.G. Lowe: Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision, Vol. 60, No. 2, Nov. 2004, pp. 91 – 110.

[2] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool: Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding, Vol. 110, No. 3, June 2008, pp. 346 – 359.

[3] M. Calonder, V. Lepetit, C. Strecha, P. Fua, BRIEF: Binary Robust Independent Elementary Features, 11th European Conference on Computer Vision, Heraklion, Greece, 05- 11 Sept. 2010, Part IV, pp. 778 – 792.

[4] J. Matas, O. Chum, M. Urban, T. Pajdla: Robust Wide-Baseline Stereo from Maximally Stable Extremal Regions, Image and Vision Computing, Vol. 22, No. 10, Sept. 2004, pp. 761 – 767..

[5] C. Arteta, V. Lemptisky, J.A. Noble, A. Zisserman: Learning to Detect Cells using Non-Overlapping Extremal Regions, 15th International Conference on Medical Image Computing and Computer-Assisted Intervention, Nice, France, 01-05 Oct. 2012, pp. 348 – 356.

[6] D. Martinec, T. Pajdla: Consistent Multi-View Reconstruction from Epipolar Geometries with Outliers, 13th Scandinavian Conference on Image Analysis, Halmstad, Sweden, 29 Jun-02 July 2003, pp. 493 – 500.

[7] R. Kimmel, C. Zhang, A.M. Bronstein, M.M. Bronstein: Are MSER Features Really Interesting?, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 33, No. 11, Nov. 2011, pp. 2316 – 2320.

[8] S. Varah, N. Grujić: Target Detection and Tracking using a Parallel Implementation of Maximally Stable Extremal Region, NVIDIA GPU Technology Conference, San Jose, CA, USA, 19 March 2013.

[9] F. Kristensen, W.J. MacLean: Real-Time Extraction of Maximally Stable Extremal Regions on an FPGA, IEEE International Symposium on Circuits and Systems, New Orleans, LA, USA, 27-30 May 2007, pp. 165 – 168.

[10] E. Salahat, H. Saleh, A. Sluzek, M. Al-Qutayri, B. Mohammad, M. Ismail: A Maximally Stable Extremal Regions System-on-Chip for Real-Time Visual Surveillance, 41st Annual Conference of the IEEE Industrial Electronics Society, Yokohama, Japan, 09-12 Nov. 2015, pp. 2812 – 2815.

[11] V. Petrović, J. Popović-Bozović: Towards Real-Time Blob Detection in Large Images with Reduced Memory Cost, 3rd International Conference on Electrical, Electronic and Computing Engineering, Zlatibor, Serbia, 13-16 June 2016, pp. EKI2.2 1 – 6.

[12] R. Sedgewick, K. Wayne: Algorithms, Addison-Wesley Professional, Upper Saddle River, NJ, USA, 2011.

[13] R. Szeliski: Image Alignment and Stitching: A Tutorial, Foundations and Trends® in Computer Graphics and Vision, Vol. 2, No. 1, Jan. 2006, pp. 1 – 104.

[14] P.H.S. Torr, A. Zisserman: MLESAC: A New Robust Estimator with Application to Estimating Image Geometry, Computer Vision and Image Understanding, Vol. 78, No. 1, Apr. 2000, pp. 138 – 156.