A Low-Power Vision System With Adaptive Background Subtraction and Image Segmentation for Unusual Event Detection

Michele Benetti, Massimo Gottardi, *Member, IEEE*, Tobias Mayr, and Roberto Passerone, *Member, IEEE*

***Abstract*— This paper presents a smart ultra-low power vision system targeted to video surveillance applications. The sensor embeds a low-level image processing technique that autonomously detects unusual events occurring in the scene, relying on adaptive background subtraction. The resulting binary image is then directly segmented by an FPGA, which triggers the higher layer of processing, transferring only aggregate feature information. The on-board processing relieves the rest of the vision system from expensive computation. The 104 × 104 pixels vision chip consumes 80 μW at 30 frames/s, while segmentation dramatically cuts down the amount of data to be transferred, resulting in an extremely low-power system suitable for embedded applications.**

***Index Terms*— Background subtraction, segmentation, low-power, video surveillance, autonomous.**

I. Introduction

THE growing interest in smart, low-power, networked sensors is attracting increasing attention in many fields of application, from battery-operated surveillance and monitoring systems in smart cities to smart human-machine interfaces in mobile devices. In this context, vision is the richest source of information describing our surrounding environment, and video monitoring applications are increasingly widespread. In order to provide immediate feedback, most video surveillance applications employ a human observer, who monitors a number of cameras for unusual events. The problem with this approach is that human observers cannot attentively follow 10 cameras for over 15 minutes, and lose 90% of their concentration capability after only 20 minutes [1]. Therefore, the probability of reacting immediately to an event captured by a surveillance camera network is estimated at 1 out of 1,000.

Automatic detection of unusual events from video sequences is therefore an important issue, and is increasingly adopted in embedded applications. However, interpreting a scene requires data-intensive vision algorithms, which, in turn, require large computing resources and high power. Scaling power in vision technology, even if limited to few use cases, is a big challenge and cannot be simply approached through the optimization of each single component, disregarding a system-level design strategy. Currently, commercial imagers are mainly targeted to high-quality image reproduction, meeting the huge demand of vision technology by the multimedia market. These sensors continuously deliver data, which often needs to be read and interpreted by a processor in real-time. Although this computing paradigm is very powerful and straightforward, it is not efficient from the energy point of view. The system has to manage a large amount of data, provided by the imager at frame rate, which is often redundant with respect to the tasks to be executed. This is especially true of those surveillance applications where the scene exhibits only slow or local changes, and the events of interest occur rarely. This turns into an oversizing of the computing resources, which represent the main limitation for the power scaling of a system. Under these assumptions, the sensor and the low-level image processing algorithm represent the most critical part of the entire system. In the last decade, significant effort has been put into the development of novel smart CMOS cameras embedding on-chip features extraction and image compression with the main goal of optimizing the computing resources and reducing the overall power consumption [2]–[8].

In this paper, we employ a custom vision sensor [9], [10] and develop the image interpretation system that detects anomalous behaviors of pixels with respect to their past history by means of an embedded robust algorithm for dynamic background subtraction and image segmentation. The sensor adaptively updates its sensitivity and thresholds in order to efficiently filter out noise and slow ambient light changes. The output of the vision sensor is a binary image where the asserted pixels represent a potential anomalous event. An efficient segmentation algorithm clusters the active pixels, extracting a set of features that characterize the scene in terms of its

Manuscript received April 16, 2018; revised June 29, 2018; accepted July 12, 2018. Date of publication August 14, 2018; date of current version October 2, 2018. This paper was recommended by Associate Editor X. Zhang. *(Corresponding author: Roberto Passerone.)*

M. Benetti is with Dyson Technology Ltd., Malmesbury SN16 0RG, U.K. (e-mail: michele.benetti@dyson.com).

M. Gottardi is with the Integrated Radiation and Image Sensor Research Group, 38100 Trento, Italy (e-mail: gottardi@fbk.eu).

T. Mayr was with the Department of Information Engineering and Computer Science, University of Trento, 38123 Trento, Italy. He is now with Max Valier Technology High School, 39100 Bolzano, Italy (e-mail: tobiasmayr89@gmail.com).

R. Passerone is with the Department of Information Engineering and Computer Science, University of Trento, 38123 Trento, Italy (e-mail: roberto.passerone@unitn.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2018.2857562

overall dynamics and distribution. This results in a robust image compression, with a significant reduction of the data to be delivered to and processed by the downstream high-level algorithms for scene analysis and interpretation. These functions are typically run by an embedded processor, which can operate in sleep mode most of the time, and potentially activate higher resolution imagers when an unusual event is detected. With ability to operate only when necessary, our main contribution is a careful design at the system levels, employing a simplified standard segmentation algorithm, that leads to a large reduction of the power consumption of the system, making it suitable for embedded battery-operated surveillance applications.

The paper is organized as follows. After reviewing the related work in Section II, Section III presents the overall system requirements and architecture, discusses the principles of operation and explores the details of the implementation. Then, Section IV presents the experimental setup, and then discusses the performance of the implemented system both in terms of its functionality (background subtraction and image segmentation), and in terms of the achievable data rates (worst case and typical). Finally, Section V concludes the paper.

II. Related Work

Advances in standard CMOS Active Pixel Sensors (APS) have facilitated the development of high-performance imagers, with reduced pixel pitch, large spatial resolution and high miniaturization level. High dynamic-range cameras and high frame-rate imagers certainly represent some of the most relevant examples of sub-micron CMOS technology [11]–[13]. However, with the constant increase of spatial resolution, the imager has to dispatch large amount of data at high speed to sustain the video frame rate. Recent trends toward wireless sensor networks necessitate an efficient way to extract visual data from a camera meeting the limited energy budget of the sensor node. Conventional scanned imagers are not able to fulfill these requirements due to their poor efficiency in the use of the signal bandwidth and the requirement for expensive video processing on the raw pixel data. For this reason, it is important to use sensors that collect only the necessary information in a scene and are able to process data in advance. In the literature, several examples of vision sensor implementations are reported, targeted to low-power applications [14], [15]. In this work, we employ a custom sensor with embedded dynamic background subtraction [9], and develop a dedicated filter and segmentation algorithm to extract and deliver only the information required for surveillance applications. In particular, we focus on the segmentation aspect of the system, and its integration with our imager. The development of algorithms and digital architectures for image segmentation has been widely addressed in the literature, and different approaches have been proposed (see, in particular, the excellent review by He *et al.* [16]). Here, we are concerned with segmentation for unusual event detection, rather than for object recognition, and therefore opt for a simpler solution that minimizes both hardware and energy requirements.

To further increase energy efficiency, imagers have been implemented as array of pixels where each pixel handles its own information and dispatches data by means of an event-based asynchronous protocol, called address-event representation (AER) [17], [18]. This is in contrast to frame-based processing, where all frames must be entirely processed pixel-by-pixel, even if they do not contain any significant data. An event-driven approach for clustering events associated with moving objects was proposed by Litzemberger *et al.* [19]. While several implementation exploit desktop PCs for event-based data processing, an interesting embedded event-based system eDVS was used within robotic applications [20]. Such system showed a power consumption as low as 23 mW. Nevertheless, computation on event-based sensor data may become extensive considering the random order, either spatial and temporal, of the high amount of events detected [21]. Hence, one shortcoming of purely asynchronous sensors is that the processing unit must be able to handle the peak rate of asynchronous events, which can be orders of magnitude larger than the average. This is problematic in our specific application, especially since the random order would require maintaining the full history of the image and entail a large number of comparison to detect the connected components. In practice, as our system delivers only the active blob information, which corresponds to events in the image, the combination of the imager and embedded data processing behaves in a way similar to the asynchronous representation, with a better fit in terms of memory and computational constraints.

Several methods exist to segment and find connected regions in an image, also called *blobs*, and to assign them a unique label. We can distinguish two fundamental approaches. In *raster-scan* algorithms, the image is scanned in raster direction, from top to bottom and from left to right. A new label is associated to each pixel that is not connected to other pixels in the same row or in the row above [22]. Otherwise, the pixel is associated to the same label of the (previous) pixel it is connected to. When two blobs are disconnected in one row, but "merge" in the next row, a label conflict (or label equivalence) arises. This problem can be solved by performing multiple forward and backward raster scans [23], propagating equivalent labels. Methods have been developed to solve the problem with only two scans [24], or even with only one scan [25], by recording the required information on the fly.

Contrary to raster scan methods, in *search-and-label* algorithms [26] the image is searched for unlabeled pixels. When one is found, a new label is assigned to the pixel and is propagated transitively to all neighboring pixels. This step is repeated until all objects are labeled. While the search algorithm might scan the image in raster direction, the label propagation requires in general an irregular access to the image. Contour-tracing algorithms [27] operate in a similar fashion, but they first look for border pixels of objects, detecting outer and inner contours, and then label the rest of the component. Also contour-tracing algorithms read the image in an irregular way.

Several methods have been introduced recently to improve image recognition algorithms by refining the concept of "object" [28] and super-pixel [29], by operating directly on compressed video sequences [30] or by jointly segmenting multiple images to search for common objects [31], [32].

In this work, we favor a simpler approach oriented towards providing sufficiently condensed data about the clusters in the scene that can be used to determine whether to wake up higher-level, and more energy demanding processing steps in the context of surveillance applications. In particular, we adopt the raster-scan algorithm for several reasons. First, the sensor delivers the image pixel by pixel in the usual direction, matching the raster-scan order. In addition, this algorithm requires that we buffer only one row of the image for processing, making it a good fit for an implementation on a board with limited resources, and reducing the processing latency. The low memory requirements also limit both static and dynamic power consumption. One difficulty of the raster-scan algorithm is the label equivalence resolution, which involves more complex operations, especially with the label memory. However, the particular application we are targeting does not require perfect label equivalence resolution, because we are not interested in getting the exact shape of objects. Instead, in the context of surveillance and alarm triggering, it is more important to know how large the percentage of changing areas is, compared to the whole scene, while still taking only larger connected objects and discarding the smaller ones.

To achieve our goal, we implement the segmentation and labeling procedure on an FPGA. Of particular interest, and closer to our approach, Appiah *et al.* [33] demonstrate the use of an FPGA to both model and remove the background and label the connected components. In our work, background subtraction is embedded directly in the analog sensor pixel structure, resulting in extremely low power operation. For the labeling function, we do take inspiration from [33] and adopt a run-length encoded strategy. Our implementation, however, is at the same time simpler, avoiding the use of an external RAM, and integrates filtering during run-length encoding, instead of adopting a more costly morphological structuring element. Several other methods have been reported in the literature using FPGAs [25], [34]–[41]. We will discuss them in detail later in Section IV. Relevant to our work, Han *et al.* [42] have proposed an FPGA implementation of image segmentation using Support Vector Machines. While this approach may be more flexible, owing to its learning capabilities, the performance is not yet sufficient to work in real time. At the same time, Zhou *et al.* [43] propose a generic method to increase the efficiency of image segmentation, in terms of both computation time and memory, by performing a coarse segmentation on a downsampled image, and then refining the result. While the method can be applied to several different algorithms, this approach requires having the whole image available, and is more appropriate to a software-based implementation with high resolution sensors. Rusci *et al.* [44] propose the use of a dedicated low-power processor in place of the FPGA. However, the results are extrapolated from an FPGA-based emulator, as the processor silicon implementation is not yet available. In addition, the design employs a simpler imager which computes contrast information rather than performing background subtraction. Finally, we should mention that the FPGA can also be employed to integrate background subtraction. For instance, Rodriguez-Gomez *et al.* [45] propose a low-resource approach using the Codebook algorithm, which stores information for each pixel regarding its variance.

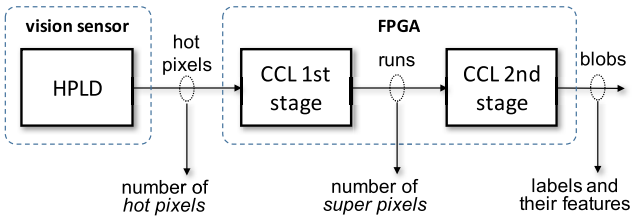


Fig. 1. System architecture. The first processing step (HPLD) is executed by the vision sensor. The other two steps are accomplished by the FPGA: the CCL-1 stage calculates the *super-pixels*; the CLL-2 stage performs the labeling.

III. System Architecture

To achieve high efficiency, the system architecture follows a hierarchical approach, where the lower processing layers progressively distill the information and wake up the higher layers of the detection algorithm only when needed, as shown in Figure 1. The system is composed of two parts. The first is a custom vision sensor embedding an adaptive background subtraction algorithm at the pixel level. The output of the sensor is a binary image, where the asserted pixels, called *hot pixels*, identify those pixels that differ from the background, denoting a potentially anomalous situation. The second level of processing is implemented on an FPGA, and is used to cluster the hot pixels into coherent connected components, called *super-pixels*, and to extract their features, condensing the information to be delivered to higher level processing layers, typically handled by a processor. The horizontal arrows show the data that is exchanged from one stage to the next, while the vertical arrows denote the kind of information that can be extracted from the different stages.

At the level of the vision sensor, background subtraction is intended to filter out all those situations in which nothing abnormal is happening on the scene, and identify the unusual events only. To make the detection reliable, the algorithm must adapt to both slow changes (such as natural illumination), and noise. At the second level, the standard Connected-Component Label (CCL) algorithm [46] further increases the robustness of the event detector. In practice, one can determine if an unusual event is present in the scene according to the following criteria:

1) the total number of hot pixels delivered by the sensor;
2) the number of *connected-component super-pixels* delivered by the first part of the CCL algorithm;
3) the labels and their features which are computed during the second part of the CCL algorithm.

The robustness of the decision taken on the basis of the number of hot pixels alone is affected by their sparsity. Pixels can be asserted either near the edges of moving shapes or after a sudden variation in the illumination of the scene. In both cases, the pixels take a certain number of frames to switch back to the inactive state, as the sensor adapts to the new conditions. The number of frames can differ from pixel to pixel, making adjacent pixels become inactive at different frames, producing

thick borders near the edges in case of a moving object, as well as a sort of salt-pepper binary noise distributed in the scene caused by ambient light variations. These affect the total number of hot pixels in subsequent frames reducing the reliability of the minimum allowable threshold used to detect the anomalous event.

To minimize the impact of these effects, the first step of the CCL algorithm erodes sparse hot-pixels and provides Connected-Component Super-Pixels (CCSP), as required by the second criterion. In this phase, the information on the number of CCSP is more robust but requires that information regarding previous lines be kept in memory. The second step of the CCL provides Connected-Component labels and their features can be evaluated in order to take application-dependent decisions. More specifically, in our implementation the sensor can detect variations both above and below the background. It is thus possible to obtain a pair of different CCLs, sometimes referred as Double Connected-Components Labels [47]. In the rest of this section we discuss these two processing levels in more detail.

A. Vision Sensor and Background Subtraction

The first stage of the system makes use of a custom integrated circuit that combines the sensor with the low-level processing layer [9], [10]. Background subtraction is performed by executing the *Hot Pixel Learning and Detection* (HPLD) algorithm [48]. HPLD works by keeping track of two reference images, each storing for each pixel a *minimum* and a *maximum* value, denoting a range of "quiescent" background conditions. Every pixel P_i of the current image is compared to the corresponding pixels min_i and max_i of the reference images to determine whether the pixel is *cold*, i.e., it lies within the normal reference range, or *hot*, i.e., it deviates from them and is therefore in the foreground. To account for slowly changing lighting conditions and to filter out noise, the reference images are also updated after every acquisition to dynamically adapt to the environment. The update procedure works by moving the reference values in the direction of the current frame. Hence, if $P_i \geq min_i$, then min_i is incremented; conversely, if $P_i < min_i$, then min_i is decremented. The update rule of the maximum value max_i is accomplished with the complementary logic. The amount of the increment or decrement can be adjusted depending on the specific application.

The low-level processing uses analog components which directly interface to the sensors. The two frame buffers needed to store the reference images are integrated in every pixel. This makes the update and comparison operations fast, and simple to implement with a classic SIMD (Single Instruction Multiple Data) architecture. For every frame, the pixel light intensity V_P is compared with the two dynamic thresholds V_{Max} and V_{Min}, stored in the frame buffers, and which take into account the pixel past behavior. The status of the pixel is denoted by two output bits Q_{Max} and Q_{Min}. As long as V_P lies within the two thresholds, no unusual event is detected (*cold-pixel*), and the two bits are 0. Trespassing one of the two thresholds, the pixel is recognized as anomalous and one of the two bits, either Q_{Max} or Q_{Min} depending on which threshold was violated, is asserted (*hot-pixel* $= Q_{Max} + Q_{Min}$). After the pixel readout, the two thresholds are updated according to the result of the detection phase.

Unlike the update step of the original HPLD algorithm, the two memories are not updated by adding or subtracting a fixed value, which would be too complex to be implemented in CMOS. Instead, a running average has been adopted, where the update value linearly depends on the difference between the values of the current pixel and the related memory through a defined coefficient α, called *learning rate*. In the implemented algorithm, the threshold which has been trespassed by V_P is updated quickly, while the one which was not involved is updated more slowly. This is accomplished by using *two* learning coefficients, α_H for the "hot side", and α_C for the "cold side", implemented as switched-capacitor low-pass filters. In practice, different learning rates can be obtained by clocking the filter once every n frames, instead of every frame. This solution is particularly simple, as no analog component is involved and requires a digital control block placed at the periphery of the imager. Under steady-state conditions, both thresholds slowly converge toward the current signal V_P, minus a small offset to avoid triggering on signal noise, retrieving the maximum pixel sensitivity and waiting for an intensity change to occur. More formally, the update steps apply the following update rules:

- High Threshold - *hot-pixel* ($V_P > V_{Max}$):

$$V'_{Max} = V_{Max} + \alpha_H(V_P - V_{Max}); \tag{1}$$

- High Threshold - *cold-pixel* ($V_P \leq V_{Max}$):

$$V'_{Max} = V_{Max} + \alpha_C(V_P - V_{Max}); \tag{2}$$

- Low Threshold - *hot-pixel* ($V_P < V_{Min}$):

$$V'_{Min} = V_{Min} + \alpha_H(V_P - V_{Min}); \tag{3}$$

- Low Threshold - *cold-pixel* ($V_P \geq V_{Min}$):

$$V'_{Min} = V_{Min} + \alpha_C(V_P - V_{Min}). \tag{4}$$

where the primed variable represents the updated value. Figure 2 [9] shows the change of V_{Max} (top red line) and V_{Min} (bottom blue line) as V_P (dashed line) changes. By means of these moving thresholds, the pixels absorb slow brightness changes, as well as fast ones, if they are repetitive and regular, such as noise, as shown at the end of the plot. Faster, irregular (unusual) changes are instead revealed by the pixels.

An example of the functionality of the sensor is shown in Figure 3. When no motion is detected, the sensor does not deliver hot pixels. As soon as a person enters the scene, some pixels are asserted by a change in light intensity, becoming hot, shown in white in the figure. Hot pixels have some persistence, which can be programmed according to the specific application, and will take several frames to be absorbed, and become cold pixel again, after the person has gone. The number of hot pixels is usually a small amount of the total number of pixels of the sensor, and can range between 10% and 20% of the total resolution in typical people monitoring applications [9]. This value is much larger in Figure 3, because the person is placed very close to the camera. After detection,

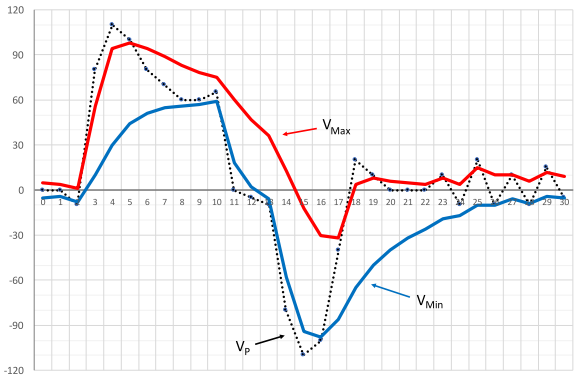


Fig. 2. Graphic representation of the adaptive background-subtraction implemented in every pixel of the array ([9], Figure 2). The zone between the two threshold curves (V_{Max}, V_{Min}) represents a range of values inside which the pixel (V_P) is considered normal (*cold-pixel*).

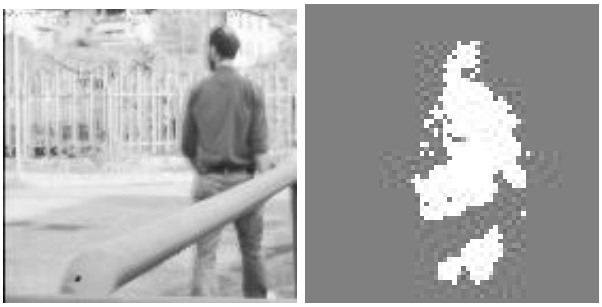


Fig. 3. Example of background subtraction executed by the sensor: a) image acquired by the sensor showing a walking person; b) related binary image provided by the sensor. White pixels represent the moving part of the image.

the hot pixels are assembled into a binary image which is further analyzed by *image segmentation*, which identifies the connected components of hot pixels, described next.

B. Segmentation

The interface between the sensor and the main processor is implemented on an FPGA, which reads out the raw image data and performs image segmentation and labeling. The FPGA is a Xilinx Spartan 6 mounted on an Opal Kelly XEM6001 board. The board is used to program the FPGA, as well as to read out the FPGA memory and transfer the values to a PC, in order to be able to visualize the images.

To reduce memory requirements, we compress the data using run-length encoding while looking for connected pixels in the current row, using the run information of the previous row to detect overlaps [33]. Comparing runs of the current and the previous row, rather than pixels, reduces the computational complexity, even considering the overhead required to compute the run-length encoding itself, as several pixels are handled at the same time. The system block diagram is shown in Figure 4. It consists of three main blocks that implement the interface to the imager and the host processor, run-length encoding and blob detection. The memory that stores the features and their labels is also instantiated here. The next sections discuss each of the blocks and their interactions in detail.

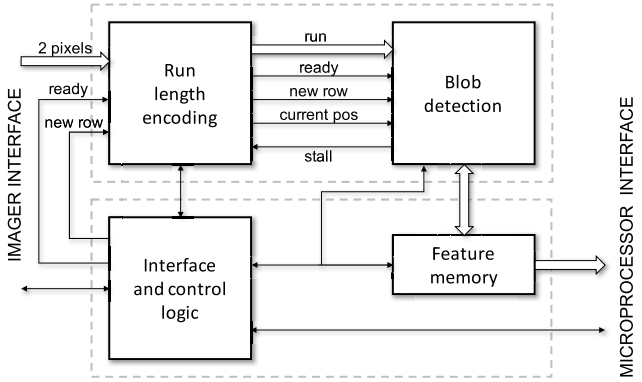


Fig. 4. System block diagram. The Run Length Encoder receives the pixel data and forwards run data to the Blob Detection module, which stores the feature information in memory. Interface and control logic orchestrates the whole behavior.

1) Run-Length Encoding (RLE): The Run Length Encoding (RLE) stage, shown on the top left in Figure 4, receives two pixels from the imager as input, together with two signals indicating when the data is ready and the end row condition. The module outputs the run-length code of the incoming row, and also informs the next stage when a new row begins.

During this step, the algorithm performs a simultaneous run-length encoding and filtering of the image to remove the salt and pepper noise. More specifically, a sequence of active pixels is considered part of the foreground run only if its length is larger than a minimum value FG_{min}. Symmetrically, the length of sequences of background pixels must be at least BG_{min}. This obviously affects the computation of the runs, which makes use of two counters (fg-counter and bg-counter) to keep track of the consecutive foreground and background pixels along the row. The algorithm work as follows. The counter are initially set to zero. Upon detecting the first foreground pixel, the fg-counter is incremented, and the position of the pixel is stored. After this, the fg-counter is incremented for any consecutive foreground pixel. When a background pixel is found, and if the fg-counter has reached FG_{min}, the bg-counter is incremented until another foreground pixel is detected. At this point, there are two possibilities:

- if bg-counter is at least BG_{min}, then a run was detected. Its end position is stored, the counters are reset to zero, and the block resumes looking for a new run;
- otherwise, the value of bg-counter is added to fg-counter, effectively elevating the pixels to the foreground, and the algorithm continues counting the consecutive foreground pixels as before.

The same operations are performed at the end of the row. Thus, the algorithm rejects runs of foreground and background pixels which are shorter than their minimum length. The selection of these parameters largely depends on the particular features of the scene to be analyzed. In our experiments below, we will use conservative values to avoid detecting single-pixel blobs.

Each time a new run is detected, the start and end positions are passed to the blob detector for further processing. The end position corresponds to the position *after* the last pixel of

the run. This way, the two positions are never the same, even if the run is only one pixel long (if allowed by FG_{min}), and a zero end-position can be used to mark an invalid run, for instance at the end of the row.

While no run is "active", i.e., the last run was already delivered, and no new run was found since then, the RLE stage also outputs the current position in the row. This way, the BD stage can fast-forward to the next run in the row-buffer, when it finds that the currently loaded run lies behind the position provided by the RLE.

Run-length encoding clearly reduces the memory requirement. If storing the raw image, one needs to store the value of the pixel and the value of the label assigned to the pixel. Assuming 7 bits per label, with a resolution of 104×104 pixels, a total of $104 \cdot (7 + 1) = 832$ bits per row are needed. Run-length encoding reduces this amount. Setting the minimum run length FG_{min} to 3 pixels, with BG_{min} equal to 1 to separate the runs by at least 1 background pixel, the maximum number of runs in a row is

$$\frac{\#px}{FG_{min} + BG_{min}} = \frac{104}{3+1} = 26. \tag{5}$$

Each run has a start and an end position, which require 7 bits each ($\lceil \log_2(104) \rceil = 7$). Thus, the number of bits needed to store information about all runs in a row (including the label to which they have been assigned) takes at most

$$26 \cdot (14 + 7) = 546 \tag{6}$$

bits, saving approximately one third of storage space. Computing the RLE may be seen as additional overhead in terms of resources. However, the output of our system includes feature information, such as the bounding box. Performing run length encoding on the pixels inherently provides the information of connected blobs in a row, comparing the start and end of the run, and thus effectively performs an operation that has to be carried out anyway, resulting in no additional hardware. Furthermore, it is easy to look for connections between two rows by simply comparing the runs, instead of doing a per pixel comparison.

2) Blob Detector (BD): The Blob Detector (BD), shown on the top right in Figure 4, is the core of the segmentation algorithm. It acquires the runs identified by the RLE block, looks for overlaps, calculates features, creates new blobs or merges runs into existing blobs. The operations are supported by a row buffer, which records the runs of the previous row, and by a feature and blob memory, both stored in a block RAM.

The row buffer is laid out in the block RAM using a double buffering technique, storing odd and even rows, which alternate between current and previous row. The same RAM is used to store the blob information. The blob label also works as an index that makes it easy to find the information in the RAM. The block is configured as a "true dual-port" RAM, so that two different memory cells can be accessed at the same time, either in reading or writing mode, independently. Having two ports allows the block to load the full blob information in one step, while the run data can be stored in the buffer.

The BD stage is normally in a ready state, where it waits for a new run from RLE and monitors the current processing position in order to load the appropriate run and blob information from the previous row buffer. When a new run is forwarded, BD checks if there is an overlap with the previous row. If so, it updates the information about the blob features contained in the feature memory. Otherwise, it creates a new label, assigns it to the new run, and creates a new record in the feature memory. In all cases, the start and end position and the label of the new run are stored in the row buffer (to make it available during the computation of the next row). Access to the RAM has a latency of 2 clock cycles, one to provide the address, and another to acquire the output.

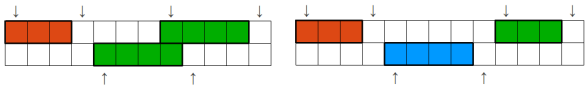


Fig. 5. Blob detection with (left) and without (right) overlap; the arrows show the stored start and end positions.

```
1: while refRun_end < thisRun_start do
2:     load next reference run
3: end while
4: if refRun_start ≤ thisRun_end then
5:     add to blob
6: else
7:     new blob
8: end if
```

Fig. 6. Run overlap detection.

Detecting an overlap or connection between foreground pixels of the "current run" and a run in the previous row, which we refer to as the "reference run", is accomplished by comparing the start and end position of the runs (see Figure 6 for pseudo-code). First, the algorithm loads reference runs from the previous row until it finds one whose end lies after the start of the current run. This ensures that an overlap is at least possible. At this point, it checks if the start position of the reference run is less than or equal to the end position of the current run. If this is the case, then an overlap is detected, and the current run is added to the blob of the reference run. Otherwise, the current run constitutes a new blob, and the information is stored in the memory. Figure 5 shows examples of runs in the two rows, with and without overlap, highlighting the start and end positions of the runs. Because the end position of a run is the position *after* its last pixel, this algorithm detects connectivity with the 8 surrounding pixels (8-pixel connectivity). It is easy to change to 4-connectivity by slightly altering the comparisons for equality in the two conditions in Figure 6.

Because of the goals of this system, it is not necessary to store the complete segmented image, with pixel-per-pixel label information. Instead, it is sufficient to collect and calculate features of the found blobs while processing the image. The features that are extracted by the algorithm are

- the bounding rectangle: left, top, right and bottom coordinates (each 7 bit wide, extended to 8 bit for simpler memory management);
- the effective number of foreground pixels in the blob (12 bit wide, extended to 16).

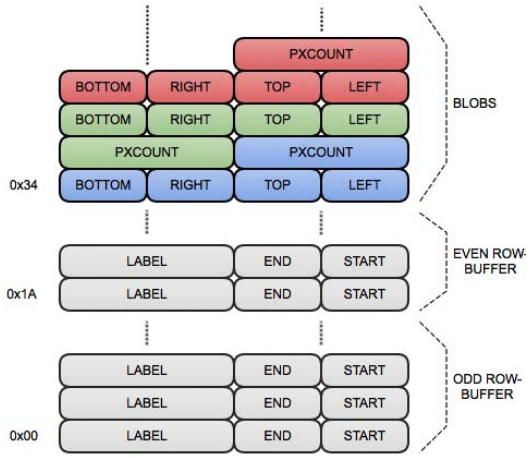


Fig. 7. Memory layout; the gray cells represent the two row-buffers, the colored cells show blob information.

Once sufficiently many blobs are detected, the higher-level processing should intervene to notify the alert and activate a more in depth analysis. For this reason, we can restrict the system to only 127 labels. In total, with 127 labels and 48 bits per label, storing the features requires 6 kbits of RAM. Figure 7 shows the memory layout, organized in 512 cells of 32 bits each (16 kbit). For comparison, the full 104×104 binary image would instead require approximately 10 kbits, while the complete labeled image (one label per pixel) would require 74 kbits. The run-length encoded and labeled image, assuming a worst case of 26 runs per row, would still require up to 55 kbits. In the average case, the savings in terms of memory are much larger.

3) Interface and Control Logic: The Interface and Control block provides the clock and global reset signals to the segmentation module, as well as all the signals required to interface the FPGA to the imager and to the host PC, via a USB connection.

The RLE and BD blocks are logically connected in a pipeline. Because the processing in the BD stage may not always require the same number of clock cycles, that stage can stall the RLE stage. In that case, the RLE still continues to grab and elaborate the data that it receives from the sensor, but when it is ready to deliver data to the BD, it stores that data and waits until the blob detector is done with processing and ready for the new data. Differently, the RLE stage cannot stall the sensor, because the acquisition can not be delayed. Because the run length encoding needs to be active during the whole time of image acquisition, while the blob detection has to wait for a new run to finish, the control logic stalls the blob detector during the wait intervals, reducing the amount of consumed power.

While the RAM is dual port, both ports are used by the BD stage during operations. Thus, the interface logic can disable the RLE and BD stage, and relay the RAM ports towards the PC interface. In particular, the interface logic notifies the PC by communicating the number of blobs that were found in the last image, and then transfers the contents of the RAM to the PC 16 bits at a time. These functions are implemented through standard interface block provided by the Opal Kelly board.

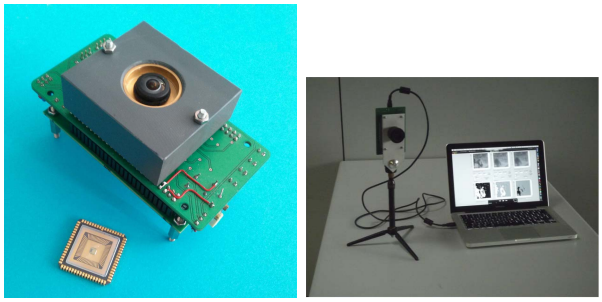


Fig. 8. System prototype based on the vision sensor.

IV. Experimental Results

In this section, we show the results of the combined background removal and image segmentation algorithm, and discuss the advantages compared to a traditional setup.

A. Experimental Setup

The system prototype, shown in Figure 8, consists of the sensor mounted on the Opal Kelly board, connected to the PC. For the purpose of demonstration, the images that the sensor acquires are stored in the FPGA memory, such that they can be transferred to the PC. Nonetheless, the goal of this system is not to display images. However, it is impractical to look at the bare blob information to check whether the algorithm works as expected. Thus, we visually integrated the blob information in the real time image that is displayed on the PC, showing the bounding rectangles. To achieve this, the program reads out the label information from the FPGA RAM and takes the information about the corners of each blob. This information is then used to draw a rectangle with OpenCV appropriate functions. When the software reads out the raw image produced by the sensor, it checks whether the sensor is acquiring an odd or even frame, and reads out the frame on which the sensor is not currently working. This is necessary to avoid simultaneous write access by the acquisition logic and read access by the PC through the board USB interface.

In a real use case, the pixel data would simply be discarded, and only the high level information would be transferred. Thus, in a real application, after the segmentation, some decision logic would decide if the next stage of processing, for example a microcontroller which detects motion of the blobs, should be triggered. This could be decided for example on the basis of blob sizes. To pass on the information to the next stage, the blob information needs to be read out from RAM. In order to be able to test this feature, and also to verify the correctness of the algorithm results, we integrated a readout function into the PC software controlling the test board.

B. Background Removal and Segmentation

The result images that are shown in Figure 9 to 12 were obtained by filming a moving hand at 30 fps in front of the vision sensor, equipped with a wide-angle lens. Since the sensor preforms the online background subtraction, only moving parts of the scene are visible. Thus, only movements that involve *intensity changes* on pixels are tracked. Notice that

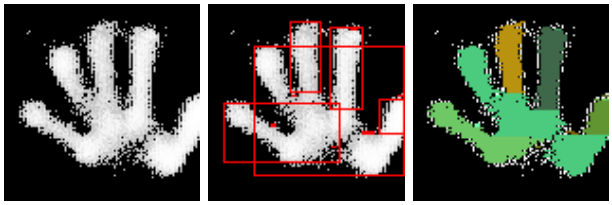


Fig. 9. Rapid appearance of the hand, fully visible; left: original image from the sensor; center: detected blobs; right: labeled image.

Fig. 10. The hand is waving and the palm is vanishing; original image - detected blobs - labeled image.

Fig. 11. After waving, only the fingers remain visible; original image - detected blobs - labeled image.

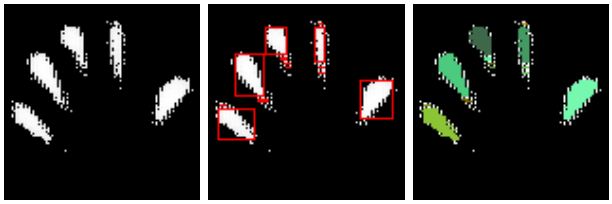


Fig. 12. Original image - detected blobs - labeled image.

in a hand waving motion, the palm usually does a rotating movement, thus no movement perpendicular to the edges is registered by the sensor in this area. This explains why, with the exception of the first image (Figure 9), only the fingers are visible.

We show three images for every example. The first always shows the original image delivered by the sensor, after the inherent background subtraction. The middle frame shows the same image again, overlaid with the bounding rectangles of the blobs. The third image shows the fully labeled image, where different colors represent the different blobs. This image is not returned by the system, but is shown here simply to get a better impression of what connected components were identified by the algorithm. These results demonstrate the correct operation of the algorithm.

A second experiment consists of a freely moving pattern displayed on a PC monitor. The pattern is a black star moving from right to left on a white background. In this experiment,

Fig. 13. Experimental results of the proposed image processing flow on real images. a) image acquired and HLPD processed by the sensor; b) super-pixels obtained by the first part of the CCL algorithm; c) labels obtained by the last part of the CCL algorithm.

we distinguish between the two thresholds. Therefore, the left-side edges are black *hot-pixels*, detecting light-to-dark variations (V_{Max}), while the right-side edges are white *hot-pixels*, detecting dark-to-light variations (V_{Min}). Unlike the previous set of experiments, which showed the original images, in this case we show the partly processed images, where isolated pixels have been filtered out by the algorithm.

Figure 13 shows three snapshots referring again to the three phases of the proposed image processing. Figure 13.a) shows the binary image processed by the sensor. It is possible to detect several isolated pixel mostly located at the edges of the moving pattern. At this stage, the total number of pixel contributes to the first-level FPGA wake-up. Figure 13.b) shows the super-pixel image, obtained by running the first part of the CCL algorithm, corresponding to run-length encoding and filtering. The number of super-pixels can be used to trigger the second part of the CCL algorithm. Figure 13c) finally shows the identified blobs, together with their bounding box. This is the last processing stage which is application-independent. Now, the criteria adopted here to proceed with the labeling phase depends on the targeted application and can be based on different features, such as blob area or blob aspect ratio.

C. Resource and Performance Estimation

Table I compares the resource requirements and the performance of several implementations of connected component analysis found in the literature with those used in our prototype for a Spartan 6 device, highlighted in bold. The table indicates the kind of algorithm that is used (mainly CCL with a few exceptions) and the number of passes on the image. A fair comparison is made difficult because of the use of different devices, as listed in the table, with fairly different characteristics. Nevertheless, the second section of the table shows the requirements in terms of resources: look-up-tables (LUTs), fabric registers and block RAMs (BRAM). In the last case, we report the number of bits, as the size of the BRAMs differs for different devices. The third section reports the achieved performance, in terms of clock frequency and frame rate (FPS). The algorithms employed are generally suitable for images of arbitrary size. The table reports the image size that corresponds to the resource usage, and in some cases the largest number of blob labels that can be handled. When no value is indicated, the implementation is able to

TABLE I

COMPARISON OF THIS WORK WITH EXISTING FPGA SOLUTIONS FOR CONNECTED COMPONENT ANALYSIS. CCL: CONNECTED COMPONENT LABELING, BS: BACKGROUND SUBTRACTION, NF: NOISE FILTERING, MUM-SH: MUMFORD-SHAH, MS: MEAN-SHIFT, N/A: NOT AVAILABLE

Reference	Lee [34]	Johnston [25]	Ma [35]	Ito [36]	Appiah [33]	Zhao [37]	Klaiber [38]	**This work**	Ngan [39]	Zhang [40]	Trieu [41]
Year	2007	2008	2008	2010	2010	2013	2016	**2018**	2011	2015	2015
Algorithm	CCL	CCL	CCL	CCL	CCL-BS	CCL	CCL	**CCL-NF**	CCT	Mum-Sh	MS
N. of passes	2	1	1	1	2	1	1	**1**	2	–	2
Device	n/a	Spartan II xc2s200	Virtex II xc2v6000	Stratix ep1s25	Virtex IV xc4vlx160	Virtex II xc2v3000	Kintex 7 k325t-2l	**Spartan 6 xc6slx16**	Stratix II 2s60	Virtex 7 vc707	Virtex IV xc4vlx160
LUTs	n/a	1,364	1,757	1,596	1,232	4,587	493	**1,595**	2,680	31,315	32,800
Registers	n/a	585	600	–	1,316	3,154	296	**383**	3,564	22,186	–
BRAM (bits)	External	40,960	72,000	53,760	342,000	234,000	108,000	**16,000**	1,984,470	27,648,000	3,384,000
Clock (MHz)	n/a	25	40.63	72.66	65	95.7	185.59	**48**	50	100	138.8
Resolution	640x480	640x480	640x480	2048x2048	640x480	256x256	256x256	**104x104**	60x200	512x512	768x512
N. of labels	255	256	4,096	– –	– –	– –	– –	**127**	– –	– –	– –
FPS	200	60	100	17	35	100	2,265	**30**	50 - 120	n/a	30

handle all possible labels. In none of the reported cases was data concerning power consumption available in the literature. Background subtraction is only included in the implementation by Appiah *et al.* [33].

Lee *et al.* [34] were first to propose an FPGA implementation for connected component labeling. In their approach, they store the whole image acquired from a frame grabber and binarized using a simple threshold. The algorithm performs a double scan to identify the labels. Details on the implementation are however not available. Johnston and Bailey [25] develop a single scan implementation, using a merger table to store the labels for conflict resolution. Limiting the design to 256 labels reduces the memory requirements. The same authors improve on their design by relieving the limitation on the number of labels, with a modest increase of the required memory, achieved by recycling labels between rows [35]. Subsequently, Ito and Nakano proposed a method using a single scan to approximate the result, focusing on reducing the latency to obtain the first label [36]. A parameter is used to control the amount of look-ahead for label merging. Its value strongly affects the resource requirements. Table I reports the results with zero look-ahead, to match the characteristics of our solution. We have already discussed the work by Appiah *et al.* [33, Sec. II], which uses run-length encoding to process the image. Zhao *et al.* [37] also use a run-based labeling approach. Their implementation is somewhat more complex, leading to a larger resource usage than comparable solutions. More recently, Klaiber *et al.* [38] propose an efficient architecture to reduce the amount of required memory. The algorithm employs dedicated memories to record patterns and speed up the analysis, for images of different sizes, up to 8 K. Here we report the resource usage for the smallest image.

Our proposed system differs compared to previous approaches for the integration with low-power background subtraction, and for the use of run-length encoding to simultaneously filter out noise and reduce the amount of data to process. The overall requirements are modest, considering the kind of device we are using, and generally in line or below those reported in the state of the art. In particular, we process the image as it is acquired, avoiding the need to store it in memory, and favor a simpler approach which results in very low memory requirements, since we do not employ a merger table. Without any particular optimization, the system runs at 48 MHz on a low-cost device. The 30 fps value reported in the table is due to the imager. With a worst case latency of 3 clock cycles per run, the segmentation system can sustain the operation in real time also for large sensors running at high frame rate.

Besides the CCL implementations, Table I also reports the data for more complex algorithms. Ngan *et al.* [39] present an FPGA implementation of the Connected Component Tree (CCT) algorithm, focusing in particular on optimizing the data structure to use memory efficiently. Our approach differs significantly, as we do not need the full complexity of CCT for our analysis and we employ a binarized instead of a gray scale image, computing the result as the image is acquired. Zhang *et al.* report an implementation of the Mumford-Shah regularization algorithm, used for image segmentation as well as reconstruction for tomography applications. This is, in absolute terms, the most complex solution, and is not meant to run in real time. More recently, Trieu and Maruyama [41] have proposed an FPGA implementation of the mean-shift algorithm, suitable for segmenting color images. While the approach achieves real-time performance, its complexity requires considerable hardware resources, and is therefore not suitable for our low-power application.

The overall power consumed by the 104×104-pixel sensor was measured at 80 μW at 30 fps [10], turning into a Figure Of Merit (FOM) of 246 pW/pixel·frame, thus contributing very little to the overall energy consumption of the system. The chip is fabricated in a 350 nm technology, has a pixel pitch of 16 μm and measures $2.5 \cdot 2.5$ μm^2.

The absolute power consumed by the FPGA in our prototype, which is clearly much larger than the sensor, is less significant, as alternative devices could be used with considerable differences in terms of power supply (which affects power consumption quadratically), current and total capacitance.

For instance, because of the low resource usage, our implementation could easily fit in extremely low power devices, such as the Microsemi Igloo.[1] In this paper we show the proof of concept of the working system, while the development of a dedicated FPGA board employing this technology is part of our future work. Thus, in order to estimate the improvement in power consumption of the proposed approach with respect to a standard one, it makes more sense to consider instead the achievable data rate of the link between the sensor and the processor, in our case the PC.

The data rate is an indicator of how well the system performs in removing redundancy or irrelevant information and representing the information contained in the image. Thus, it is also related to the processing power needed to interpret the information in a useful way, and therefore gives a hint for the power required by the higher-level hardware in the processing chain. A low data rate means that much of the processing has been moved from the PC to the low-level hardware. In the following, we therefore compare different solutions, assuming a frame rate of 30 fps, a pixel depth of 7 bpp and a resolution of 104×104 pixels.

1) Camera With General Purpose Processor: In a conventional approach, the raw image of an ordinary camera would be transferred to the processing unit. At the greyscale precision of 7 bits per pixel, this results in a data volume of

$$7 \cdot 104 \cdot 104 = 75712 \text{ bit} \approx 74 \text{ kbit}$$

per frame, or in a data transfer-rate of

$$74 \text{ kbit} \cdot \text{frame}^{-1} \cdot 30 \text{ fps} = 2.17 \text{ Mbps}.$$

While lossy compression could reduce this rate by high factors, it is usually not allowed to have coding losses in images that need further processing, since coding artifacts could highly influence change detection algorithms. Therefore, only lossless compression methods are suitable in this case, leading to a gain of about 1 : 4 [49] at best, thus leaving a data rate of around

$$2.17 \text{ Mbps} \div 4 \approx 550 \text{ kbps}.$$

The processing system would then need to perform adaptive background subtraction, apply thresholding for binarization, do the segmentation, after having decided whether it is necessary, and decide what to do on the basis of the found blobs. All these operations would be performed on every frame, 30 times per second, most of them regardless of whether there are relevant changes in the scene, or any changes at all.

2) Custom Sensor With General Purpose Processor: The imager system would take care of the initial processing steps of adaptively subtract the background, and apply hot-pixel thresholding, leaving the processor only with the task of segmenting the image and interpreting the blobs. Regarding data transfer, this means that only binary pixels need to be transferred, resulting in

$$1 \text{ bpp} \cdot 104 \cdot 104 = 10816 \text{ bit} \approx 10.5 \text{ kbit}$$

or

$$10816 \text{ bit} \cdot \text{frame}^{-1} \cdot 30 \text{ fps} \approx 317 \text{ kbps}.$$

[1] https://www.microsemi.com/product-directory/fpgas/1689-igloo#overview

This already represents a significant reduction in data, compared to traditional systems, especially considering that for binary images, compression like run-length encoding could efficiently decrease the data rate even more, without adding much computational overhead. Furthermore, we need to take into account that with this setup the general purpose processor can stay in a low power state, as long as the sensor detects fewer than a critical number of hot pixels. This, together with the implementation of the background subtraction algorithm being integrated in hardware and potentially ultra low-power, contributes to considerable power savings.

3) Full Proposed Architecture: In the proposed system, information redundancy is further reduced. The segmentation logic takes the binarized image, finds connected blobs and calculates their features. The data rate necessary to encode this information depends on the type and number of features that are extracted. In the test setup, the bounding rectangle corner coordinates and the pixel count in the blob were considered. This results in

- 7 bits per coordinate, adding up to 28 bits;
- 14 bits for the pixel count.

So 42 bits are needed to encode information about one blob. At most, in a 104×104 pixel image,

$$26 \text{ runs} \cdot \text{row}^{-1} \cdot \frac{104 \text{ rows}}{2} = 1352 \text{ blobs}$$

can be found. However, the number of blobs is usually much lower. A reasonable size, below which the blob can be rejected, is around 1% of the total image size (see for example Figure 11, where the index finger is represented by the smallest blob, still of a size of $10 \cdot 10$ pixels). Therefore, the maximum number of blobs needed to be transferred would be limited to about 135. This equals to a (uncompressed) data rate of

$$135 \text{ blobs} \cdot 42 \text{ bits} \cdot \text{blob}^{-1} = 5670 \text{ bit},$$
$$5670 \text{ bit} \cdot \text{frame}^{-1} \cdot 30 \text{ fps} \approx 166 \text{ kbps}.$$

On average, the number of blobs is typically much lower. Considering a reasonable number of 20 blobs per frame, the data rate is further reduced to

$$42 \text{ bits} \cdot \text{blob}^{-1} \cdot 20 \text{ blobs} \cdot \text{frame}^{-1} \cdot 30 \text{ fps} \approx 25 \text{ kbps}.$$

4) Summary: To summarize, the image sensor alone already greatly reduces the amount of data because it performs background subtraction, which reduces the pixel depth from 7 to 1 and potentially leaves only the active pixels to be transferred. On top of this, the segmentation algorithm further reduces the data necessary to represent the relevant information contained in the scene, by at least a factor of 2 in the worst case, and more than 10 in the typical case. The overall gains in terms of data rate reduction are summarized in Table II. In addition to reducing the amount of transferred data, the sensor-embedded hardware system can dynamically wake up the higher levels of processing only when necessary. Therefore, by optimizing across levels, the proposed architecture dramatically reduces the operating duty cycle and consequently the power consumption.

TABLE II

COMPARISON OF DATA RATES, USED AS A FIGURE OF MERIT FOR POWER CONSUMPTION, NORMALIZED WITH RESPECT TO PIXEL DEPTH, FRAME RATE AND RESOLUTION, TAKING A CONVENTIONAL VISION SYSTEM AS REFERENCE

System	data units	bits/unit	data rate	gain
Standard (raw)	104x104 px	7bits	2218 kbps	0.25
Standard (compressed)	104x104 px	7bits	550 kbps	1
Sensor only	104x104 px	1bit	317 kbps	1.7
Full (worst case)	135 blobs	42bits	166 kbps	3.3
Full (typical case)	20 blobs	42bits	25 kbps	22

V. CONCLUSIONS

We have presented a micropower vision sensor and low-level processing intended to be part of an image analysis and interpretation algorithm. The architecture is based on a robust low-level adaptive background subtraction, which is implemented at the pixel level, taking advantage of analog fully parallel architecture. The binary image delivered by the sensor is directly segmented using efficient digital techniques, implemented on an FPGA. The prototype and experimental results demonstrate the correct behavior of the system and provide estimates of the potential power savings for higher level processing algorithms, which utilize only aggregate feature data. Our current work includes the development of an FPGA board employing dedicated low-power components. At the same time, we are studying the implementation of the segmentation algorithm directly on the sensor chip, thus providing an even tighter integration and reducing even further the overall power consumption. The integrated solution could then function as a standalone low-maintenance device for embedded unusual event detection. To prove the effectiveness of the system in real applications, we are developing a live demonstrator making use of a larger imager recently presented at ISCAS 2018 [50]. The integration of segmentation to the demonstrator is part of our current work.

REFERENCES

[1] C. Gomersall, *Challenges and Developments in Intelligent Video Surveillance*. Accessed: Aug. 3, 2018. [Online]. Available: http://www.sourcesecurity.com/news/articles/co-3223-ga.1852.html

[2] I. Cevik and S. U. Ay, "An ultra-low power energy harvesting and imaging (EHI) type CMOS APS imager with self-power capability," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 9, pp. 2177–2186, Sep. 2015.

[3] F. D. V. R. Oliveira, H. L. Haas, J. G. R. C. Gomes, and A. Petraglia, "CMOS imager with focal-plane analog image compression combining DPCM and VQ," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 5, pp. 1331–1344, May 2013.

[4] J. Choi, S. Park, J. Cho, and E. Yoon, "A 1.36 μW adaptive CMOS image sensor with reconfigurable modes of operation from available energy/illumination for distributed wireless sensor network," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2012, pp. 112–113.

[5] M. T. Chung and C. C. Hsieh, "A 0.5 V 4.95 μW 11.8 fps PWM CMOS imager with 82 dB dynamic range and 0.055% fixed-pattern noise," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2012, pp. 114–116.

[6] J. Fernandez-Berni, R. Carmona-Galan, and Á. Rodriguez-Vazquez, "Ultralow-power processing array for image enhancement and edge detection," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 59, no. 11, pp. 751–755, Nov. 2012.

[7] N. Cottini, L. Gasparini, M. De Nicola, N. Massari, and M. Gottardi, "A CMOS ultra-low power vision sensor with image compression and embedded event-driven energy-management," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 1, no. 3, pp. 299–307, Sep. 2011.

[8] K. Ito, B. Tongprasit, and T. Shibata, "A computational digital pixel sensor featuring block-readout architecture for on-chip image processing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 1, pp. 114–123, Jan. 2009.

[9] N. Cottini, M. Gottardi, N. Massari, R. Passerone, and Z. Smilansky, "A 33 μW 64×64 pixel vision sensor embedding robust dynamic background subtraction for event detection and scene interpretation," *IEEE J. Solid-State Circuits*, vol. 48, no. 3, pp. 850–863, Mar. 2013.

[10] M. Benetti, M. Gottardi, and Z. Smilansky, "A 80 μW 30 fps 104×104 all-nMOS pixels CMOS imager with 7-bit PWM ADC for robust detection of relative intensity change," in *Proc. IEEE ESSCIRC*, Sep. 2013, pp. 303–306.

[11] M. Mase, S. Kawahito, M. Sasaki, Y. Wakamori, and M. Furuta, "A wide dynamic range CMOS image sensor with multiple exposure-time signal outputs and 12-bit column-parallel cyclic A/D converters," *IEEE J. Solid-State Circuits*, vol. 40, no. 12, pp. 2787–2795, Dec. 2005.

[12] A. Xhakoni and G. Gielen, "A 132-dB dynamic-range global-shutter stacked architecture for high-performance imagers," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 6, pp. 398–402, Jun. 2014.

[13] A. Spivak, A. Belenky, A. Fish, and O. Yadid-Pecht, "A wide-dynamic-range CMOS image sensor with gating for night vision systems," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 2, pp. 85–89, Feb. 2011.

[14] F. D. V. R. Oliveira, J. G. R. C. Gomes, J. Fernández-Berni, R. Carmona-Galán, R. del Río, and Á. Rodríguez-Vázquez, "Gaussian pyramid: Comparative analysis of hardware architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 9, pp. 2308–2321, Sep. 2017.

[15] A. Rodríguez-Vázquez, R. Carmona-Galán, J. Fernández-Berni, V. Brea, and J. A. Leñero-Bardallo, "In the quest of vision-sensors-on-chip: Pre-processing sensors for data reduction," *Electron. Imag.*, vol. 2017, no. 11, pp. 96–101, 2017.

[16] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognit.*, vol. 70, pp. 25–43, Oct. 2017.

[17] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proc. IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct. 2014.

[18] G. Alberini, M. Gottardi, N. Massari, and R. Passerone, "A low-power clock-less pulse width modulator architecture for smart imaging," *J. Low Power Electron.*, vol. 14, no. 1, pp. 118–128, 2018.

[19] M. Litzenberger *et al.*, "Embedded vision system for real-time object tracking using an asynchronous transient vision sensor," in *Proc. 4th IEEE Signal Process. Edu. Workshop*, Sep. 2006, pp. 173–178.

[20] G. R. Müller and J. Conradt, "A miniature low-power sensor system for real time 2D visual tracking of LED markers," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2011, pp. 2429–2434.

[21] T. Delbruck and M. Lang, "Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor," *Frontiers Neurosci.*, vol. 7, pp. 223:1–223:7, Nov. 2013. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2013.00223, doi: 10.3389/fnins.2013.00223.

[22] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognit.*, vol. 42, no. 9, pp. 1977–1987, 2009.

[23] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Comput. Vis. Image Understand.*, vol. 89, no. 1, pp. 1–23, 2003.

[24] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 749–756, May 2008.

[25] C. T. Johnston and D. G. Bailey, "FPGA implementation of a single pass connected components algorithm," in *Proc. 4th IEEE Int. Symp. Electron. Design, Test Appl.*, Jan. 2008, pp. 228–231.

[26] Y. Shima, T. Murakami, M. Koga, H. Yashiro, and H. Fujisawa, "A high-speed algorithm for propagation-type labeling based on block sorting of runs in binary images," in *Proc. 10th Int. Conf. Pattern Recognit.*, Jun. 1990, pp. 655–658.

[27] F. Chang, C.-J. Chen, and C.-J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Comput. Vis. Image Understand.*, vol. 93, no. 2, pp. 206–220, 2004.

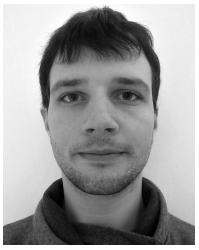
[28] H. Wang, D. Huang, K. Jia, and Y. Wang, "Hierarchical image segmentation ensemble for objectness in RGB-D images," *IEEE Trans. Circuits Syst. Video Technol.*, to be published, doi: 10.1109/TCSVT.2017.2776220.

[29] Y. Zhang, X. Li, X. Gao, and C. Zhang, "A simple algorithm of superpixel segmentation with boundary constraint," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 7, pp. 1502–1514, Jul. 2017.

[30] L. Zhao, Z. He, W. Cao, and D. Zhao, "Real-time moving object segmentation and classification from HEVC compressed surveillance video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 6, pp. 1346–1357, Jun. 2018.
[31] H. Li, F. Meng, Q. Wu, and B. Luo, "Unsupervised multiclass region cosegmentation via ensemble clustering and energy minimization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 5, pp. 789–801, May 2014.
[32] F. Meng *et al.*, "Constrained directed graph clustering and segmentation propagation for multiple foregrounds cosegmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 11, pp. 1735–1748, Nov. 2015.
[33] K. Appiah, A. Hunter, P. Dickinson, and H. Meng, "Accelerated hardware video object segmentation: From foreground detection to connected components labelling," *Comput. Vis. Image Understand.*, vol. 114, no. 11, pp. 1282–1291, 2010.
[34] D. R. Lee, S. H. Jin, P. C. Thien, and J. W. Jeon, "FPGA based connected component labeling," in *Proc. Int. Conf. Control, Automat. Syst.*, Oct. 2007, pp. 2313–2317.
[35] N. Ma, D. G. Bailey, and C. T. Johnston, "Optimised single pass connected components analysis," in *Proc. Int. Conf. Field-Program. Technol.*, Taipei, Taiwan, Dec. 2008, pp. 185–192.
[36] Y. Ito and K. Nakano, "Low-latency connected component labeling using an FPGA," *Int. J. Found. Comput. Sci.*, vol. 21, no. 3, pp. 405–425, 2010.
[37] F. Zhao, H. Z. Lu, and Z. Y. Zhang, "Real-time single-pass connected components analysis algorithm," *EURASIP J. Image Video Process.*, vol. 2013, no. 1, p. 21, 2013.
[38] M. J. Klaiber, D. G. Bailey, Y. O. Baroud, and S. Simon, "A resource-efficient hardware architecture for connected component analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 7, pp. 1334–1349, Jul. 2016.
[39] N. Ngan, E. Dokladalova, M. Akil, and F. Contou-Carrère, "Fast and efficient FPGA implementation of connected operators," *J. Syst. Archit.*, vol. 57, no. 8, pp. 778–789, 2011.
[40] W. Zhang *et al.*, "FPGA acceleration by asynchronous parallelization for simultaneous image reconstruction and segmentation based on the Mumford-Shah regularization," *Proc. SPIE*, vol. 9600, p. 96000H, Sep. 2015.
[41] D. B. K. Trieu and T. Maruyama, "Real-time color image segmentation based on mean shift algorithm using an FPGA," *J. Real-Time Image Process.*, vol. 10, no. 2, pp. 345–356, 2015.
[42] L. Han, Z. Yue, and X. Guo, "Image segmentation implementation based on FPGA and SVM," in *Proc. 3rd Int. Conf. Control, Automat. Robot. (ICCAR)*, Apr. 2017, pp. 405–409.
[43] Y. Zhou, T.-T. Do, H. Zheng, N.-M. Cheung, and L. Fang, "Computation and memory efficient image segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 1, pp. 46–61, Jan. 2018.
[44] M. Rusci, D. Rossi, M. Lecca, M. Gottardi, E. Farella, and L. Benini, "An event-driven ultra-low-power smart visual sensor," *IEEE Sensors J.*, vol. 16, no. 13, pp. 5344–5353, Jul. 2016.
[45] R. Rodriguez-Gomez, E. J. Fernandez-Sanchez, J. Diaz, and E. Ros, "Codebook hardware implementation on FPGA for background subtraction," *J. Real-Time Image Process.*, vol. 10, no. 1, pp. 43–57, Mar. 2015.
[46] M. B. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected-component labeling for arbitrary image representations," *J. ACM*, vol. 39, no. 2, pp. 253–280, 1992.
[47] J. Xiong, T. M. Nguyen, and Q. M. J. Wu, "FPGA implementation of blob recognition," in *Proc. Can. Conf. Comput. Robot Vis. (CRV)*, May 2011, pp. 125–131.
[48] Z. Smilansky, "Miniature autonomous agents for scene interpretation," U.S. Patent 7 489 802, Feb. 10, 2009.
[49] D. A. Clunie, "Lossless compression of grayscale medical images: Effectiveness of traditional and state-of-the-art approaches," *Proc. SPIE*, vol. 3980, pp. 74–84, May 2000.
[50] Y. Zou, M. Gottardi, and M. Perenzoni, "Live demostration: Low power vision sensor with robust dynamic background rejection," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, May 2018, p. 1.

Michele Benetti received the M.S. degree in material engineering from the University of Ferrara in 2005 and the Ph.D. degree in information and communication technology from the University of Trento in 2012. He was a Researcher with FBK, Trento, Italy, and CEA LETI, Grenoble, France. He is currently an Embedded Electronic Engineer with Dyson Technology Ltd., Malmesbury, U.K. His research interests span sensors, and embedded and edge computing for sensing applications related to consumer electronics.

Massimo Gottardi (M'96) received the M.Sc. degree in electrical engineering from the University of Bologna, Italy, in 1987. Since 1990, he has been doing research on CMOS vision sensors design with the Integrated Radiation and Image Sensors Research Group, Fondazione Bruno Kessler, Trento, Italy. He has co-authored three patents on optical position sensors, vision sensors, MEMS interfaces, and has over 100 papers in refereed journals and international conferences. His current research interests include energy-aware vision sensors, hardware-oriented algorithms for sensor embedded image processing, low-power computational imaging, memristor-based architectures for adaptive sensory systems for wireless sensor networks, human–machine interfaces, and Internet of Things. He has served as a reviewer for several international journals.



Tobias Mayr received the B.Eng. and M.Eng. degrees in information and communication technologies from the University of Trento, Italy, in 2011 and 2013, respectively. During his studies he focused on image processing, writing both his bachelor's and master's thesis on topics from his research field. He is currently a High School Professor of computer science and computer networking with the Max Valier Technology High School, Bolzano, Italy. Before starting to work as a Teacher, he was the Lead Programmer for several software projects at Stillalive Studios, Innsbruck, Austria.

Roberto Passerone (S'96–M'05) received the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley, in 1997 and 2004, respectively. He is currently an Associate Professor of electronics with the Department of Information Engineering and Computer Science, University of Trento, Italy. Before joining the University of Trento, he was a Research Scientist with Cadence Design Systems. He has published numerous research papers on international conferences and journals in the area of design methods for systems and integrated circuits, formal models, and design methodologies for embedded systems, with particular attention to image processing and wireless sensor networks. He has participated in several European projects on design methodologies, including SPEEDS, SPRINT and DANSE, and he was the Local Coordinator for ArtistDesign, COMBEST, and CyPhERS. He has served as the Track Chair of the real-time and networked embedded systems with ETFA from 2008 to 2010, and the general chair and the program chair for various editions of SIES.