Journal of Signal Processing Systems 52, 75–94, 2008
© 2007 Springer Science + Business Media, LLC. Manufactured in The United States.
DOI: 10.1007/s11265-007-0100-7

An Embedded Real-Time Surveillance System: Implementation and Evaluation

FREDRIK KRISTENSEN, HUGO HEDBERG, HONGTU JIANG, PETER NILSSON AND VIKTOR ÖWALL
CCCD, Department of Electrical and Information Technology, Lund University, Box 118, 221 00, Lund, Sweden

Received: 1 March 2007; Accepted: 5 June 2007

Abstract. This paper presents the design of an embedded automated digital video surveillance system with real-time performance. Hardware accelerators for video segmentation, morphological operations, labeling and feature extraction are required to achieve the real-time performance while tracking will be handled in software in an embedded processor. By implementing a complete embedded system, bottlenecks in computational complexity and memory requirements can be identified and addressed. Accordingly, a memory reduction scheme for the video segmentation unit, reducing bandwidth with more than 70%, and a low complexity morphology architecture that only requires memory proportional to the input image width, have been developed. On a system level, it is shown that a labeling unit based on a contour tracing technique does not require unique labels, resulting in more than 50% memory reduction. The hardware accelerators provide the tracking software with image objects properties, i.e. features, thereby decoupling the tracking algorithm from the image stream. A prototype of the embedded system is running in real-time, 25 fps, on a field programmable gate array development board. Furthermore, the system scalability for higher image resolution is evaluated.

Keywords: hardware, FPGA, real-time, surveillance, segmentation, morphology, labeling, tracking, image features, embedded system, video processing

1. Introduction

The demands on video surveillance systems are rapidly increasing regarding parameters such as frame rate and resolution. Furthermore, with an ever increasing data rate and number of video streams, an automated process for extracting relevant information is required. Due to the large amount of input data and the computational complexity of the algorithms, software implementations are not sufficient to sustain real-time performance for a reasonable resolution. In this paper we present an automated digital surveillance system running on an embedded platform in real-time. Algorithms that are well suited for hardware implementation with streamlined dataflow are chosen and dedicated hardware accelerators have been developed. The presented hardware platform has been developed with the goal of presenting a proof of concept for the surveillance system and to identify computational and memory bottlenecks. Furthermore, when proposing modifications to the original algorithms extensive simulations are needed, especially if long-term effects in the video sequences can be envisioned. Utilizing a reconfigurable platform based on a field programmable gate array (FPGA) reduces the simulation and development time considerably.

A conceptual overview of the surveillance system is shown in Fig. 1. The camera feeds the image processing system with a real-time image stream of

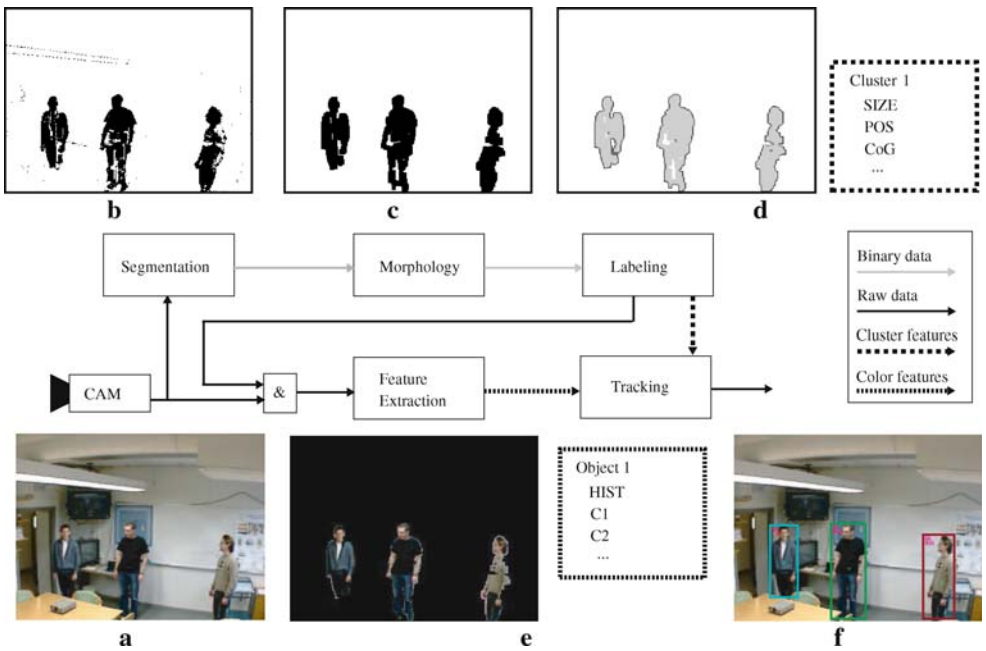

Figure 1. Surveillance system, **a** original image, **b** binary motion mask, **c** morphologicaly filtered motion mask, **d** Labeled clusters and cluster features, **e** detected objects and color features, and **f** tracking results. All tracked objects are marked with a uniquely colored frame as long as the object is visible.

25 frames per second (fps), Fig. 1a. A segmentation algorithm, in this case based on a Gaussian Mixture background Model (GMM), preprocesses the image stream and produces a binary mask in which background and foreground are separated, Fig. 1b. In theory, only the moving parts of an image should be distinguished as independent objects in the binary mask. However, in reality the mask will be distorted with noise and single objects are shattered. In order to remove noise and reconnect split objects, morphological operations are performed on the mask, Fig. 1c. These morphological operations will produce a frame of connected clusters which have to be identified, i.e. labeled. The labeled clusters together with extracted cluster features, e.g. size and position, are seen in Fig. 1d. Foreground objects, which have been cut out from the original frame, with corresponding color features is shown in Fig. 1e. In the final image, tracked objects are identified by uniquely colored Bounding Boxes (BB), Fig. 1f.

The main bottleneck of image processing algorithms, are the high memory requirements that is imposed on the hardware system both in terms of number of bits and bits per second, i.e. size and bandwidth. In this work, bandwidth reduction has primarily been addressed in the segmentation unit, through wordlength reduction and by identifying and removing redundant information. The goal with the morphological unit has been to create a data path unit that does not require any intermediate storage of the image. Both decomposition and simple Structuring Elements (SE) have been explored to reach this goal. In the labeling unit, the main issue has been to decrease the amount of data stored on-chip. Here, both carefully choosing algorithm and system level considerations have lead to a reduced memory size. Finally, the dependency between image resolution and memory requirements for all parts of the system has been investigated in order to find the constraints of a future higher resolution system.

Sections 2 to 5 present the individual blocks of the system, as outlined in Fig. 1. Each block has been implemented as a stand-alone block, but has been verified using a software simulation model of the complete system. Section 6 discusses how the individual blocks have been integrated on an FPGA board. Hardware utilization, system optimizations and system bottlenecks are also discussed in this Section. Finally our conclusions are drawn in Section 7.

1.1. Systems of Today

Intelligent surveillance is an expansive field which can be seen from the increasing number of products commercially available on the market. Both surveillance cameras and larger systems with advanced image analysis capabilities are emerging. Three of the largest actors on the market are AXIS Communications, Sony and IBM.

AXIS Communications is one of the global market leaders in network video products and have specialized in professional network video solutions for remote monitoring and security surveillance [1]. Features of AXIS surveillance cameras include built-in motion detectors and WLAN modules. Several subsections of a scene can be specified for motion detection, each with an individual sensitivity level. However, the detection is as for most embedded video motion detection algorithms, basic and based on frame difference.

One of the most advanced surveillance cameras on the market today is Sony's SNC-CS50 [2]. According to the specifications both advanced motion detection and unattended object detection can be performed by the camera, however not simultaneously. The unattended object detector reacts if an object is left in one place for more than a specified duration and the motion detection is based on the last 15 frames in order to reduce noise sensitivity. However, a live demonstration showed that the camera reacts slowly to motion and is sensitive to light changes.

IBM has recently released the Smart Surveillance System (S3) release-1 to end customers on a pilot basis. Compared to the previously mentioned products, S3 is by far the most advanced. However, S3 is not designed to be used in an embedded camera but as a separate software system to which several cameras are connected. According to the homepage [3], the system is capable of object detection that is insensible to light and weather changes as well as camera shake. Detected objects can be both tracked and classified, typical classification labels include, person, group, and vehicle. In addition to real-time tracking and classification, all detected events are stored alongside the original data stream for fast event based searching in the captured videos. Since no live demonstrator is available and the current release is limited to a small number of test users, it is not possible to evaluate the claimed capability of the system.

From the above overview it is seen that there is a huge gap between the capabilities of the embedded surveillance cameras, AXIS and Sony, and the large scale surveillance system, IBM. A similar trend can be seen in academia, either large systems implemented in software or isolated algorithms implemented in dedicated hardware are published. For example, W4 [4] is a system that, in addition to motion detection and tracking of multiple people on monocular gray-scale video, tries to detect activities such as persons carrying objects and different body postures. Other surveillance systems that both track and classify objects are found in Stauffer and Grimson [5] and Collins et al. [6]. The former focus on classifying events like people and cars arriving and leaving through a co-occurrence matrix and the latter describes an attempt to monitor a complex area using a distributed network of cameras. A more recent system that track multiple humans in complex situations is Zhao and Nevatia [7], were people are tracked in 3D using an ellipsoid shape model. In addition, motion modes, e.g. walking, running, and standing, and body posture are estimated. For a more extensive survey of visual surveillance we refer to Hu et al. [8]. Common for all of these systems is that they are, or need to be, executed on one or more general purpose computers in order to reach real-time performance with an image resolution of 320×240 or more. Most published hardware implementations deal with smaller parts of a surveillance system, e.g. implementation of motion segmentation, image filtering, or video codec. Some examples are Aguilar-Ponce et al. [9] and Fahmy et al. [10] that describe the implementation of a motion segmentation algorithm and a high speed median filter, respectively. FPGA implementations of video codecs for MPEG-4 and H.264 are found in Schumacher et al. [11] and Kordasiewicz et al. [12].

The proposed system, in this paper, is trying to bridge this gap by taking some of the functionality from the software system and move it into the camera. To have the functionality inside the camera instead of running it on a separate computer has some obvious benefits. Most importantly, the amount of data that has to be transmitted over the network can be reduced, especially important if a wireless scenario is considered. For larger installations this could be critical, e.g. at airports where hundreds of cameras are installed and the aggregated bandwidth

becomes substantial. The output from each of these cameras have to be routed to a security central, a reduced bandwidth could then be the difference between using the existing network or installing a completely new one. To move all functionality of a stand-alone software system into the camera will probably never be feasible. However, if some of the functionality is moved, the software system could be redesigned to use the output from the smart cameras instead of the raw image stream that is used today. In larger security systems, all cameras would then be connected through a system backbone to a central unit with a coordinating functionality, whereas in smaller systems it could be enough to install only smart cameras. Recently, another embedded image system has been presented by Philips Research labs. The system is based on two processors, one for low level image operations and one for higher level applications, connected through a dual-port memory [13–15]. However, surveillance applications have yet to be demonstrated on it and the amount of available memory limits the possibility to process color images.

The proposed system is an early attempt to move a complete hardware accelerated surveillance system onto a stand-alone embedded system, consisting of an image sensor, an FPGA with an embedded processor, and some external memories.

2. Segmentation

Over the years, various video segmentation algorithms have been proposed, e.g. frame difference, median filters [16] and linear predictive filters [17]. However, to achieve robustness in multi-modal background scenarios, an algorithm based on GMM proposed in Stauffer and Grimson [18] and Russo and Russo [19] is chosen. A GMM is required for modeling repetitive background object motion, e.g. swaying trees, reflections on a lake surface or a flickering monitor. A pixel located in the region where repetitive motion occurs will generally consist of two or more background colors, i.e. the RGB value of that specific pixel toggles over time. This would result in false foreground object detection with most other adaptive background estimation approaches.

The advantage of the GMM is achieved by using several Gaussian distributions for each pixel. The drawback is the imposed computational complexity and high memory bandwidth that prohibits real-time performance using a general purpose computer. In our simulations, a frame rate of only 4–6 fps is achieved for video sequences with a 320×240 resolution, on an AMD 4400+ dual core processor. For a real-time video surveillance system with higher resolution, hardware acceleration is required. The rest of this section will present how the GMM can be improved and efficiently implemented, for additional information we refer to Jiang et al. [20].

2.1. Algorithm Formulation

The algorithm is briefly formulated as follows: In a sequence of consecutive video frames, the values of any pixel can be regarded as a Gaussian distribution. Characterized by mean and variance values, the distribution represents a location centered at its mean values in the RGB color space. A pixel containing several background object colors, e.g. a swaying leaf on a tree in front of a road, can be modeled with a mixture of Gaussian distributions with different weights. The weight of each distribution indicates the probability of matching a new incoming pixel. A match is defined as the incoming pixel within a certain deviation from the center. In this paper, J times the standard deviation of the distribution is used as the threshold [18]. The higher the weight, the more likely the distribution belongs to the background. Mathematically, the portion of the Gaussian distributions belonging to the background is determined by

$$B = argmin_b \left(\sum_{k=1}^{b} \omega_k > H \right), \tag{1}$$

where b is the number of Gaussian distributions per pixel, H is a predefined parameter and ω is the weight. The mean, variance and weight factors are updated frame by frame. If a match is found, the parameters of the matched distribution are updated according to:

$$\begin{aligned} \omega_{k,t} &= (1-\alpha)\omega_{k,t-1} + \alpha, \\ \mu_t &= (1-\rho)\mu_{t-1} + \rho X_t \end{aligned} \tag{2}$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t), \tag{3}$$

where μ and σ^2 are the mean and variance, α and ρ learning factors, and X_t is the pixel value. For those unmatched, the weight is updated according to

$$\omega_{k,t} = (1-\alpha)\omega_{k,t-1}, \tag{4}$$

while the mean and the variance remain the same. If none of the distributions match, the one with the lowest weight is replaced by a distribution with the incoming pixel value as its mean, a low weight and a large variance.

2.2. *Color Space Transformation*

In theory, multi-modal situations only occur when repetitive background objects are present in the scene. However, this is not always true in practice. Consider an indoor environment where the illumination comes from a fluorescence lamp. A video sequence of such an environment was taken from our lab and 5 pixels picked evenly from the scene were measured over time. Their RGB value distributions are drawn in Fig. 2a and it can be seen that instead of 5 sphere like pixel distributions, the shapes of the pixel clusters are rather cylindrical. Pixel values tend to jump around more in one direction than another in the presence of illumination variations caused by the fluorescence lamp and camera jitter. This should be distinguished from the situation where one sphere distribution is moving slowly towards one direction due to slight daylight changes. Such a case is handled by updating the corresponding mean values in the original background model. Without an upper bound for the variance, the sphere describing the distribution will grow until it covers nearly every pixel in the most distributed direction, thus taking up a large space such that most of it does not belong to the distribution (A in Fig. 2b). A simple solution to work around this problem is to set an upper limit for the variance, e.g. the maximum value of the variance in the least distributed direction. The result is multi-modal distributions represented as a series of smaller spheres (B–E also in Fig. 2b). Although a back-

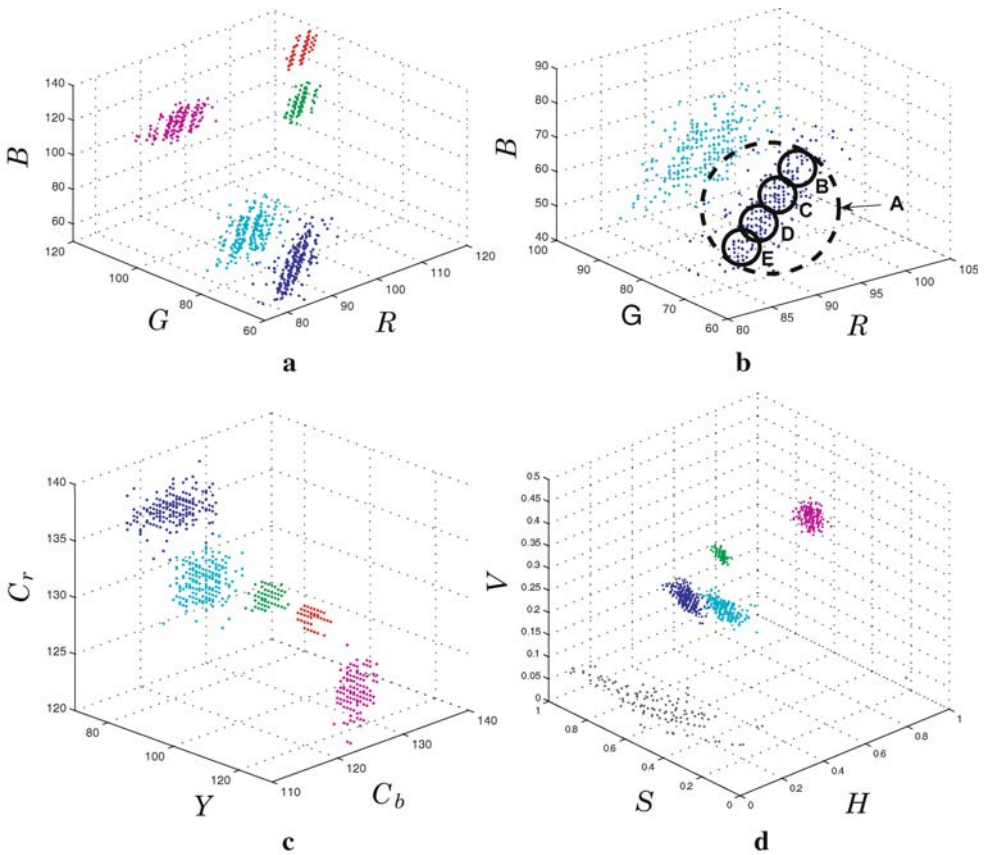

Figure 2. **a** Five distributions in the RGB color space. **b** A closer look at the 2 Gaussian distributions on the bottom in **a**. **c** Sphere distributions in the YC_bC_r space. **d** Unpredictable distributions in the HSV space.

ground pixel distribution is modeled more precisely by such method, several Gaussian distributions are inferred which are hardware costly in terms of extra parameter updating and storage.

To be able to model background pixels using a single distribution without much hardware overhead, color space transformation is employed. Both HSV and YC_bC_r space have been investigated and their corresponding distributions are shown in Fig. 2c,d. Transforming RGB into $YC_b C_r$ space results in nearly independent color components. Accordingly, in a varying illumination environment, only the Y component (intensity) varies, leaving C_b and C_r components (chromaticity) more or less independent. In Kristensen et al. [21], this feature is utilized for shadow reduction. Consequently, values of the three independent components in the YC_bC_r color space tends to spread equally and as shown in Fig. 2c most pixel distributions are transformed from cylinders back to spheres, capable of being modeled with a single distribution. The transformation from RGB to YC_bC_r is linear, and can be calculated with a low increase in computational complexity, see Section 6. On the other hand, HSV color space is no better than RGB if not worse. Unpredictable pixel clusters appeared occasionally which is hard to model using Gaussian distributions, Fig. 2d.

2.3. *Segmentation Architecture*

Maintaining a mixture of Gaussian distributions for each pixel is costly in terms of both calculation capacity and memory storage, especially at high resolution. To manage the RGB data from a video camera in real time, a dedicated hardware architecture is developed with a streaming data flow. The hardware architecture as shown in Fig. 3 is presented in Jiang et al. [20] and briefly explained as follows: A pixel value is read into the matching logic block from the sensor together with all the parameters for the mixture of Gaussian distribution from an off-chip memory and a match is calculated. In case an incoming pixel matches several Gaussian distributions, only the one with highest weight is selected as the matching one.

After the updated Gaussian parameters have been sorted, foreground detection is achieved by simply summing up the weights of all the Gaussian distributions that have a higher likelihood than the updated one. By comparing the sum with a predefined parameter H, a sequence of binary data indicating background and foreground is streamed out to the morphology block. The main bottleneck of the architecture is the high bandwidth to the off-chip memory, which will be addressed in the following.

2.4. *Wordlength Reduction*

Slow background updating requires large dynamic range for each parameter in the distributions, since parameter values are changed slightly between frames but could accumulate over time. According to Eqs. 2 and 3, the mean and variance of a Gaussian distribution is updated using a learning factor ρ. The difference of mean and variance between current and previous frames is derived from the equation as

$$\Delta_\mu = \mu_t - \mu_{t-1} = \rho(X_t - \mu_{t-1}) \text{ and}$$
$$\Delta_{\sigma^2} = \sigma_t^2 - \sigma_{t-1}^2 = \rho((X_t - \mu_t)^T(X_t - \mu_t) - \sigma_{t-1}^2). \quad (5)$$

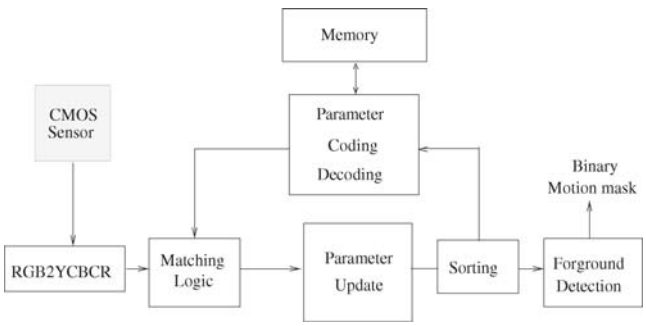

Figure 3. The architecture of the segmentation unit.

Given a small value of ρ, e.g. 0.0001, a unit difference between the incoming pixel and the current mean value results in a value of 0.0001 for Δ_μ. To be able to record this slight change, *22* bits have to be used for the mean value, where 14 bits accounts for the fractional part. Empirical results have shown that the Gaussian distributions usually are spheres with a diameter less than 10 and in this study, as well as in Stauffer and Grimson [18], $J = 2.5$. Therefore, an upper bound for the variance is set to 16 and a maximum value of Δ_μ becomes $\rho \times J \times \sigma = 0.0001 \times 2.5 \times \sqrt{16} = 0.001$, which can be represented by 10 bits. Using a wordlength lower than that, no changes would ever be recorded. In practice, the bits for the fractional parts should be somewhere in between 10–14 bits and 7–14 for the mean and variance, respectively. Together with 16 bits weight and integer parts of the mean and the variance, 81–100 bits are needed for a single Gaussian distribution. To reduce this number, a wordlength reduction scheme was proposed in Jiang et al. [20]. From Eq. 5, a small positive or negative number is derived depending on whether the incoming pixel is above or below the current mean. Instead of adding a small positive or negative fractional number to the current mean, a value of 1 or -1 is added. The overshooting caused by such coarse adjustment could be compensated by the update in the next frame. The result is that without illumination variation, the mean value will fluctuate with a magnitude of 1 which is negligible since the diameter of the Gaussian distribution is usually more than 10.

In a relatively fast illumination varying environment, fast adaptation to new lighting conditions is also enabled by adding or subtracting ones in consecutive frames. Figure 4a shows the experimental results of the coarse updating in a room with varying lighting conditions. The parameter updating

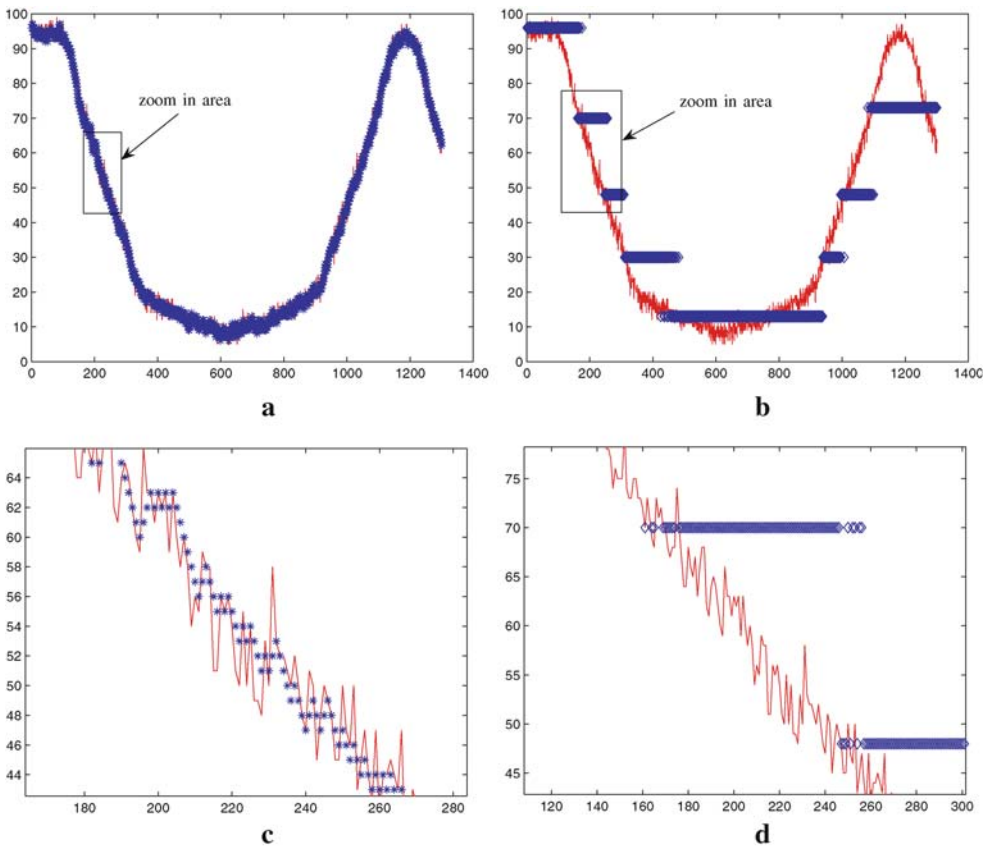

Figure 4. Parameter updating schemes comparison in fast light changing environment. One color value (*solid line*) of a RGB pixel are drawn over frames together with updated Gaussian RGB mean value (*blue diamond line*). The zoomed in area of **a** and **b** is shown in **c** and **d**, respectively.

scheme specified in the original algorithm is also drawn in Fig. 4b for comparison. A closer look at the two schemes is shown in Fig. 4c and d. From Fig. 4b and d, it is seen that parameter updating (diamond line in the figure) of the original algorithm does not work well in the presence of fast light changes. The Gaussian distribution will not keep track of the pixel value changes and Gaussian distribution replacement takes place instead of parameter updating. On the other hand, the coarse updating scheme handles such situations with only parameter updating.

With coarse updating, only integers are needed for mean specification, which effectively reduce the wordlength from 18–22 down to 8 bits. A similar approach can be applied to the variance, resulting in a wordlength of *6* bits, with *2* fractional ones. Together with the weight, the wordlength of a single Gaussian distribution can be reduced from 81–100 to only 44 bits, over *45%* reduction is accomplished. In addition, less hardware complexity is achieved since multiplication with the learning factor of ρ is no longer needed.

2.5. *Pixel Locality*

In addition to wordlength reduction, a data compression scheme for further bandwidth reduction is proposed by utilizing pixel locality for Gaussian distributions in adjacent areas. Consecutive pixels often have similar colors and hence have similar distributions. We classify "similar" Gaussian distributions in the following way: from the definition of a matching process, each Gaussian distribution can be simplified as a cube, where the center is the YC_bC_r mean value and the border to the center is specified as J times the variance. One way to measure the similarity between two distributions is to check the overlap of the two cubes. If the overlap takes up a certain percentage of both Gaussian cubes, they are regarded as "similar". The overlap is a threshold parameter that can be set to different values for different scenarios.

In the architecture, two similar distributions are treated as equivalent and by only saving non overlapping distributions together with the number of equivalent succeeding distributions, memory bandwidth is reduced. Various threshold values are selected to evaluate the efficiency for memory bandwidth reduction. With a low threshold value more savings could be achieved but at the same time more noise is generated due to increasing mismatches. Fortunately, such noise is found to be non-accumulating and can therefore be reduced by morphological filtering presented in Section 3. Figure 5 shows the memory bandwidth savings over frames with various threshold values. It can be seen that memory bandwidth savings tend to stabilize (around 50% –75% depending on threshold value) after initialization. The quality of segmentation results before and after morphology is shown in Fig. 6 where it is clear that memory reduction comes at the cost of segmentation quality. Too low threshold value results in clustered noise that would not be filtered out by the morphological filtering, Fig. 6c.

3. Morphology

As seen in the previous section, the generated binary mask needs to be filtered to reduce noise and reconnect split objects. This is accomplished by applying mathematical morphology. Erosion (ε) and dilation (δ) are the two foundations in mathematical morphology, from

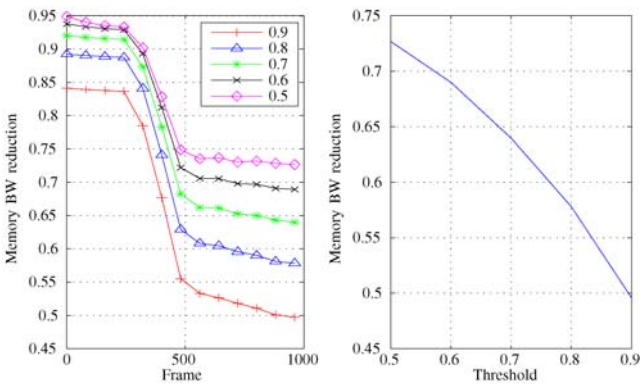

Figure 5. Memory bandwidth reduction over frames for different thresholds is shown to the *left* and memory bandwidth reduction versus threshold is shown to the *right*.

which many other extended operations are derived [22], e.g. opening, closing, and gradient. Mathematical morphology applies to many image representations [23], but only binary ε and δ is required in our system.

In an effort to make the binary morphological processing effective, a low complexity and low memory requirement architecture was proposed in Hedberg et al. [24]. This architecture has several properties and benefits which are of special interest for our application in order to easily incorporate the unit into the system. First, pixels are processed sequentially from first to last pixel. Since each operation is completed in a single image scan, a short execution time is ensured and no extra memory handling is invoked. This allows for several ε and δ units to be placed in series or parallel with only a small FIFO in between the blocks, to account for stall-cycles due to inserted boundary pixels (padding). Another property of the architecture is that the size of the SE can be changed for each frame during run-time. With a flexible SE size comes the ability to compensate for different types of noise and to sort out certain types of clusters, e.g. high and thin objects (standing humans) or wide and low objects (side view of cars).

Let I_b represent the binary input image and SE the structuring element. If SE is both reflection invariant and decomposable, i.e. $\mathrm{SE} = \widehat{\mathrm{SE}}$ and $\mathrm{SE} = \mathrm{SE}_1 \oplus \mathrm{SE}_2$, the following two equations for ε and δ can be derived

$$\begin{aligned}\varepsilon(I_b, \mathrm{SE}) &= I_b \ominus (\mathrm{SE}_1 \oplus \mathrm{SE}_2) \\ &= (I_b \ominus \mathrm{SE}_1) \ominus \mathrm{SE}_2, \end{aligned} \tag{6}$$

$$\begin{aligned}\delta(I_b, \mathrm{SE}) &= (I_b \oplus \mathrm{SE}_1) \oplus \mathrm{SE}_2 \\ &= \left(\left(I_b^{'} \ominus \mathrm{SE}_1\right) \ominus \mathrm{SE}_2\right)^{'}, \end{aligned} \tag{7}$$

where $'$ is bit inversion. Comparing Eqs. 7 and 8, it can be seen that both ε and δ can be expressed as an erosion (or as a dilation). This property is known as the duality principle. With a decomposed SE, the number of comparisons per output is decreased from the number of ones in the SE to the number of ones in SE_1 plus SE_2. However, finding decompositions to an arbitrary SE is a difficult problem and not always possible [25, 26]. In addition, for an SE to be reflection invariant it has to be symmetric in respect to both x and y axes, e.g. an ellipse. However, a common class of SEs that is both decomposable and reflection invariant is rectangles of ones. This type of SE is well suited for operations such as opening and closing, which are needed in this system. An example of ε with a decomposed SE is shown in Fig. 7, were the SE is decomposed into SE_1 and SE_2, see Eq. 6. The input is first eroded by SE_1 and then by SE_2 and the number of comparisons per output is reduced from 15 to 8.

3.1. *Morphology Architecture*

By using a rectangular SE containing only ones, ε can be performed as a summation followed by a comparison. The ε is performed by keeping track of the bits in I_b that is currently covered by the SE and are compared to its size in both the x and y direction. By decomposing the SE, the summation can be broken up into two stages. The first stage compares the number of consecutive ones in I_b to the width SE_1 and outputs a one if this condition is fulfilled. The second stage sums the result from the first stage for each column and compares it to the height of SE_2. If both these conditions are fulfilled, the output at the coordinate of the SE origin is set to one, else zero.

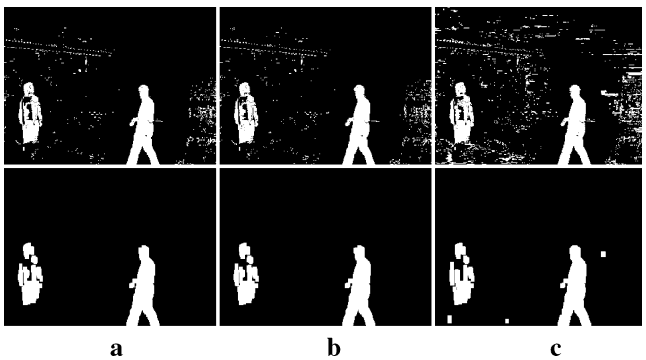

Figure 6. The result before and after morphological filtering for different thresholds, **a** original result, **b** with 0.8, and **c** with 0.4 threshold.

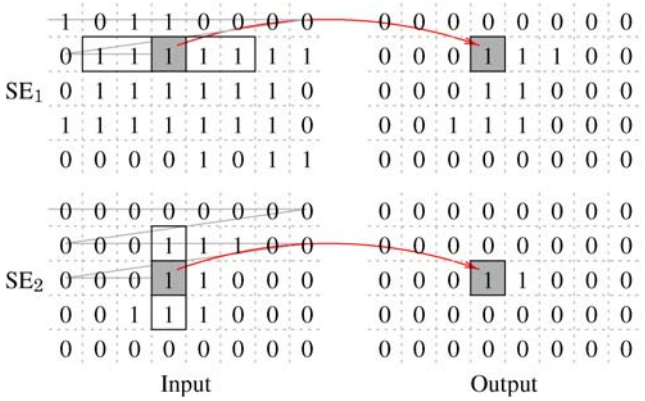

Figure 7. Input and output to an erosion were a SE of size 3×5 is decomposed into $\mathrm{SE}_1 = 1 \times 5$ and $\mathrm{SE}_2 = 3 \times 1$.

The proposed architecture is based on the observations above and is shown in Fig. 8 with corresponding wordlength in each stage. Taking advantage of the duality property, the same inner kernel is used for both δ and ε; to perform δ on a ε unit simply invert the input I_b and the result, performed in Stage-0 and 3. Each pixel in I_b is used once to update the sum, stored in the flip-flop in stage-1, that records the number of consecutive ones to the left of the currently processed pixel. When the input is one, the sum is increased, else reset to zero. Each time the sum plus the input equals the width of SE_1, stage-1 outputs a one to stage-2 and the previous sum is kept. The same principle is used in stage-2 but instead of a flip-flop, a row memory is used to store the number of ones from stage-1 in the vertical direction for each column in I_b. In addition, omitted from the figure, a controller is required to handle padding and to determine the operation to be performed, i.e. ε or δ. How, and why, padding is inserted around the boundary of an image is discussed in Hedberg et al. [24].

The wordlength in Stage-0 and 3 is a single bit whereas the wordlengths in stage-1 and 2 are proportional to the maximum supported size of the SE, i.e. $\lceil \log_2(\mathrm{SE}_{width}) \rceil$ and $\lceil \log_2(\mathrm{SE}_{height}) \rceil$, respectively. Thus, the total amount of required memory to perform ε or δ is

$$mem_{tot} = \lceil \log_2(\mathrm{SE}_{width}) \rceil + \lceil \log_2(\mathrm{SE}_{height}) \rceil I_{b,col} \,\text{bits},$$

where the first part is the flip-flop in stage-1 and second part is the row memory in stage-2. As an example, with a resolution of 320×240 and a SE size of 15×15, the required amount of memory is $\lceil \log_2(15) \rceil + \lceil \log_2(15) \rceil \cdot 320 = 1.28$ kbits. The delay line implementations in Fejes and Vajda [27] and Velten and Kummert [28], with the same resolution and SE size would require $\mathrm{SE}_{width} + (\mathrm{SE}_{height} - 1) I_{b,col} = 4.50$ kbits of memory, which is ≈ 3.5 times more.

The primary morphological operation used in this system is an opening, i.e. an ε followed by a δ. Due to the pipelined nature of the architecture, the opening operation can be performed directly on the output stream from the segmentation by placing two

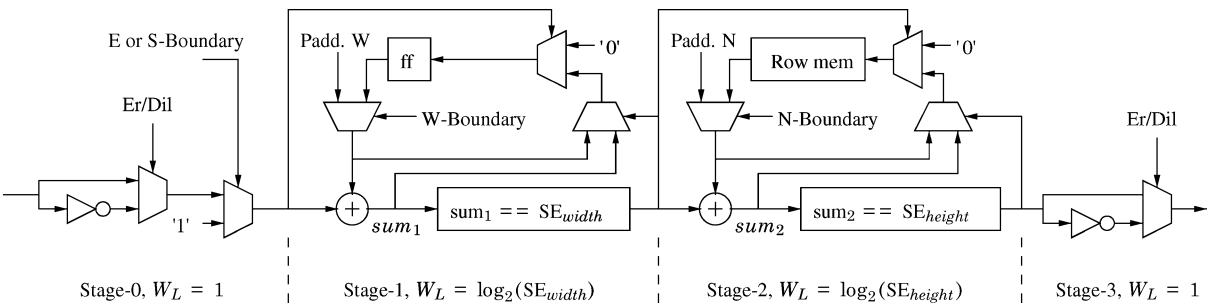

Figure 8. Architecture of the datapath in the erosion and dilation unit with corresponding wordlength (W_L) in each stage.

units in series. This will not increase the execution time but only add latency of a few clock cycles. Examples of filtered segmentation results are shown in Fig. 6. An opening operation is performed and the image is first eroded with an SE size of 5×3 and then dilated with a 7×5 SE.

4. Labeling

After the morphological operation the binary frame contains connected clusters of pixels that represent different objects of interest which should be tracked and classified. However, the system needs to be able to separate and distinguish between these clusters. Labeling has the goal of assigning a unique label to each cluster, transforming the frame into a symbolic object mask with the possibility to tie features to each cluster. Thus, labeling can be seen as the link between the clusters and their corresponding features. Labeling algorithms dates back to the early days of image processing [29] and applies to many image representations [30]. Various algorithms have been proposed over the years and a survey can be found in Kesheng et al. [31]. The algorithms can be placed into two major categories, namely

- Sequential local operations (SLO), and
- Contour tracing (CT).

The remainder of this section describes a comparison between these two types of algorithms in terms of memory requirements and which features they can extract.

In SLO based algorithms [32], a label is assigned based upon the pixels above and to the left of the current pixel which comes natural when working on streaming data. However, this type of algorithms have to solve possible label collisions. A typical label collision occurs if a u-shaped object is encountered. Scanning the image, the pillars will be assigned different labels since there is no momentary information that they are part of the same cluster. Reaching the lower middle part of the u, an ambiguity in which label to assign will occur, referred to as a label collision. A common way to solve this is to write the label collisions into an equivalence table during an initial scan and resolve them during a second. The number of label collisions per frame depends on the complexity of the cluster contours.

CT based algorithms traces and labels the contour of each cluster [33]. Labeling the contour will avoid label collisions since if a previously labeled cluster (contour) is encountered, the scan proceeds without modification. The algorithm requires a global memory scan together with additional random accesses for the CT procedure in order to label all clusters in a frame. In order to avoid pitfalls like tracing contours of possible holes inside clusters a reserved label is written on each side of the cluster. Based on this reserved label, the algorithm keeps track of whether it is currently inside a cluster or not. In the same manner, when reading the labeled result, pixels between two reserved labels can be considered part of the same cluster regardless of the pixel value. Thus, holes inside clusters can be filled which is beneficial in our application.

Both types of algorithms need a memory to store the labeled image result, mem_{label}. Due to the physical limitations of this memory, an upper bound is placed on the number of clusters that can be labeled in a frame, c_{max}. In SLO based algorithms each label collision will occupy a temporary label during the initial scan, the memory size is determined by a combination of c_{max} and the maximum number of label collisions, $l_{max,c}$. Thus, a memory overhead is introduced. In CT based algorithms the memory size is directly proportional to the image resolution and c_{max}. The memory requirement for the SLO and CT based algorithms can be written as

$$\text{mem}_{slo} = \left\lceil \log 2\left(c_{max} + l_{max,c} + 1\right)\right\rceil \cdot N \text{ bits}, \quad (8)$$

$$\text{mem}_{CT} = \lceil \log 2(c_{max} + 3)\rceil \cdot N \text{ bits}, \quad (9)$$

where N is the number of pixels in an image and $+1$ and $+3$ comes from the number of preoccupied labels.

Table 1 compiles three simulations that show the number of clusters with corresponding label collisions per frame. Sequence 1 is captured in our research laboratory, Sequence 2 is captured outdoors covering a traffic crossing, and sequence 3 is taken from the PETS database [34]. Using Eqs. 8 and 9 and figures from Table 1, SLO based algorithms would require 6, 7, 8 bits per pixel compared to CT based algorithms which would require 4, 6, 7 to handle the worst case scenario for sequence 1 to 3, respectively. Hence, CT based algorithms requires less total memory compared to SLO based algorithms to be able to label the same number of clusters.

Table 1. Three simulations on independent sequences showing the number of clusters and corresponding label collisions.

Sequence	Seq. 1	Seq. 2	Seq. 3
Mean clusters per frame	4.4	19.9	7.2
Mean label_{col} per frame	13.9	19.6	15.3
Max clusters in a frame (c_{max})	13	36	87
Max label_{col} in a frame ($l_{max,c}$)	27	36	100
Number of frames in the seq.	700	900	2500

From a system perspective, it is desirable to extract features where they have low requirements in terms of execution time and hardware complexity. Since the clusters are scanned during the labeling process, many binary features are advantageously extracted by this unit, e.g. coordinates which are used to create a BB around each cluster. The extraction procedure of many features is the same for both types of algorithms. However, a unique property of CT based algorithms is the possibility to use discrete Green's theorem during the CT phase. Green's theorem gives the relationship between a closed contour (curve) and a double integral over this cluster (plane region), enabling calculation of moments [35]. Moments can in turn be used to calculate area and center of gravity (CoG) which are important properties in this particular application, e.g. used by the tracking unit to handle occlusion.

Summarizing the comparison between the two types of algorithms, a common property is that they impose a high bandwidth together with large memory requirements. Since memory issues are the major concern in our application, arithmetic complexity in the algorithms will be traded for memory resources. Therefore, the CT based algorithm was found more suitable in our particular application and chosen for implementation, due to the following properties:

- CT based algorithms require less memory and can guarantee labeling of a predefined number of clusters.
- Both types of algorithms have the same upper bound on execution time, $t_{exe} \leq 3 \cdot (im_{height} \times im_{width})$ [36].
- CT based algorithms have the possibility to add Green's formula and thereby extract CoG,
- and have the ability to fill holes inside a cluster.

4.1. *Labeling Architecture*

An overview of the CT based architecture implemented in Hedberg et al. [36], is illustrated in Fig. 9. A FIFO is located at the input in order to stall the data stream as a frame is being labeled. The CT finite state machine (CT_{FSM}) first writes the complete frame into mem_{label}. The first and last pixel equal to 1 for this frame is marked as global start and end point respectively. After that, a second memory scan starts from the global start pixel, now also marked as local starting pixel for this particular cluster. The CT_{FSM} traces the contour of the cluster, and writes the label into mem_{label}. The CT of this cluster is completed when the local starting pixel is reached a second time. The global scan then continuous until a new cluster or the global end point is reached.

During the CT phase, the feature extraction blocks calculate, X and Y-coordinates, height, width, size, and CoG, for every cluster and stores them in the feature memory, i.e. mem_{feat}. To maximize the time the embedded SW (tracking algorithm) can access this result, a dual memory structure is used. Hence,

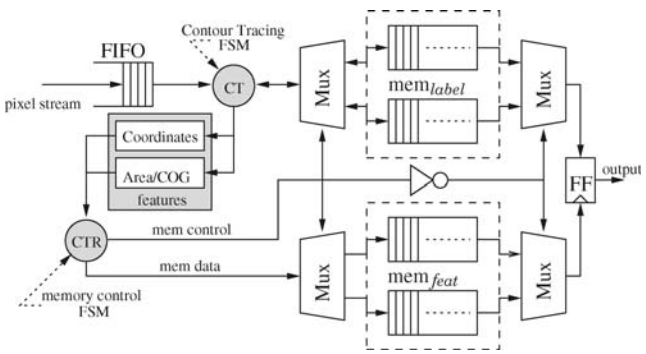

Figure 9. Overview of the implemented CT based architecture.

as the algorithm is labeling frame f in one memory pair, the tracking algorithm has access to the result of frame $f-1$ in the other memory pair. An example of a binary frame together with corresponding labeled output from the implemented architecture can be seen in Fig. 10.

Some applications do not require unique labels and binary features are sufficient. In such application, the CT based algorithm allows the label memory to be reduced to 2 bits per pixel, still maintaining correct binary feature extraction, since each cluster will get a separate entry in the feature memory. This observation is further discussed in Section 6.

5. Tracking

The goal of the surveillance system is to track persons while they remain in view of one stationary camera. Each person in view should be given a unique identity that should remain fixed even though people change place and/or disappears shortly from view. In the following text persons or things that are tracked, i.e. given a unique identity, are referred to as objects whereas objects detected by the motion detector is referred to as clusters.

Tracking of non-rigid objects, e.g. humans, is complicated and becomes even harder when it has to be performed on an embedded system with limited resources. An initial decision is hardware/software partitioning where software has the benefits of flexibility and shorter design time and the hardware has the advantage of high throughput. To take advantage of both these properties, the system is partitioned so that tasks that have to be executed directly on the image stream are implemented in hardware, while bookkeeping and conditional tasks are performed in software. The result is that tracking is performed in software and all preprocessing and calculations on the image stream are performed in hardware. The interface between hardware and software is features.

A feature is a property extracted from an object in the image, e.g. size, color, texture, or shape, that can separate different objects or recognize a certain object class. A good feature describes each object with a unique and compact code and does not change if the object is scaled, rotated, or enters an area with different lighting. This is necessary to be able to track an object through different environments, e.g. track a person standing under a lamp close to the camera who moves away towards a darker corner.

In this system there are three feature classes that are acquired from different parts of the system, at different times and during various conditions. First, cluster features acquired from the binary motion mask in the label unit. These features are calculated for each labeled cluster and for each frame. Secondly, color features are calculated if an occlusion between two objects is detected. The third feature class is prediction features that are used to make an initial guess about which objects from previous frame corresponds to which objects in the current frame. Table 2 summarize the different feature classes.

Cluster features includes minimum X and Y coordinates, height and width of the cluster, the number of pixels in a cluster (size), and CoG coordinates. These features are used as initial data to the tracking algorithm, which starts with a reconstruction phase. In this phase, objects from previous frame are reconstructed from the detected clusters. This is necessary since objects can consist of more than one cluster due to imperfect segmen-

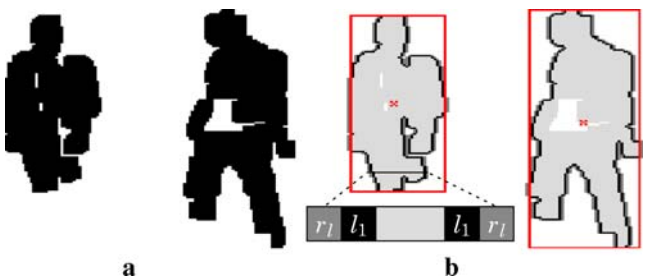

Figure 10. **a** A fragment of a typical binary input frame to the label unit, **b** corresponding labeled output. The result includes BB around each object with their CoG marked as an x. Notice r_l on each side of the cluster line segment which corresponds to the reserved label.

Table 2. The feature classes, features part of the class, and when they are calculated.

Class	Features	Calculation
Cluster	Size, min coordinates, height, width, CoG coordinates	For every cluster and frame
Color	Mean, variance, histogram	If occlusion is detected
Prediction	d'CoG, d'width, d'height, d'size	For every tracked object and frame

tation and occlusions. The reconstruction is based on the predicted position of an objects CoG and size. When two or more clusters are used to reconstruct an object, new cluster features are calculated as the weighted mean of the used clusters. Cluster features are often enough to track non-occluded objects in the video stream.

During the reconstruction phase, merges and splits are also detected. A merge occurs when two objects touch each other and become one object, i.e. an object–object occlusion, and a split is when one object becomes two objects. Both events are detected in a similar way, based on CoG coordinates and BB. The BB is defined as the minimum rectangle that completely surrounds an object or cluster and it is created with the cluster features width, height, and minimum coordinates. A merge is detected if the CoG of two tracked objects are found inside the BB of one new cluster, and a split is detected if two cluster CoG are found inside the BB of one object. An example is shown in Fig. 11.

Color features include color-mean, variance, and histogram of an object. These features have been chosen since they can be calculated from streaming data without any reordering of the pixels and produce a small number of data, i.e. minimum processing time and memory requirements. In addition, color features are size invariant and with the right color space also lighting invariant [21].

If the predicted position of an objects BB in the next frame is overlapping the predicted position of another object, i.e. an occlusion is imminent, color features are extracted and stored as a reference. During the rest of the occlusion two sets of features are extracted for each participating object, one set assumes that the object is to the right of the other object and the other set assumes that it is to the left. For example, if object A and B merge and form object C, Fig. 12 shows which parts of C that is used to calculate the feature sets for both object A and B. The four feature sets are then matched against the two stored reference sets and a left–right (LR) score is stored for each object. Depending on, if an object is best matched with the right, left or no feature set the LR score is adjusted according to

$$LR(f) = \begin{cases} LR(f-1)\alpha + K & \text{if a right match,} \\ LR(f-1)\alpha & \text{if a no match,} \\ LR(f-1)\alpha - K & \text{if a left match,} \end{cases}$$

where $\alpha < 1$ and K are constants, and f is the frame number. The larger the $|LR|$, the stronger the evidence that the object is to either right or left side, the final decision on which object is which is not taken until a split event is detected. The main advantages of this method are that no motion prediction is used to estimate the outcome and that it easily scales to more than two objects. Since no motion estimation is used, the system will not be confused if a person moves behind another person, stops, turns around and move back the same way she entered. Finally, the tracking algorithm calculates prediction features, such as motion and size predictions. Size prediction corresponds to motion prediction in the direction towards or away from the camera or an object that enters or exits the scene. The prediction is based on $g-h$ tracking [37]. Experiments to use Kalman filtering instead were

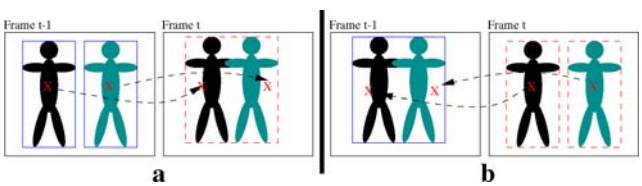

Figure 11. BBs around tracked objects are shown with *solid lines* and around new clusters with *dashed*, *x* marks the CoG. **a** Shows a merge event and **b** a split event.

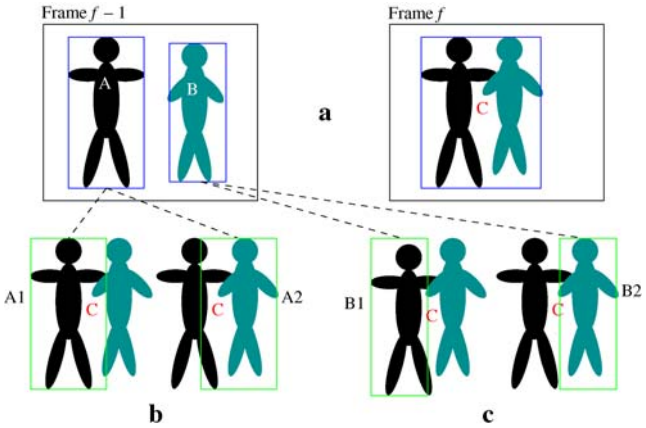

Figure 12. **a** Object A and B in frame $f-1$ and f. Object C is object A and B merged. **b** The two feature sets of A extracted from C. **c** The two feature sets of B extracted from C.

done, but the performance increase did not justify the complexity increase.

6. System Implementation and Performance

A prototype of the system is implemented on a Xilinx Virtex II pro vp30 FPGA development board, with two FPGA embedded Power PCs and a 256 MB off-chip DDR SDRAM. A KODAK KAC-9648 CMOS sensor is attached directly onto the board and is used to capture color images at 25 fps with a resolution of 320×240. The development board is shown in Fig. 13.

The architecture of the prototype is shown in Fig. 14, where black indicates custom made logic, light blue is memories and red is of-the-shelf components. The architecture is modular in the sense that each block can be replaced with other algorithms without changing the overall architecture. Modularity is achieved with independent clock domains and asynchronous FIFOs in between all major blocks. Communication between feature memories and the PPC is performed with software addressable registers and is initialized with an interrupt signal from the label unit. A custom made VGA controller makes it possible to superimpose BB around the detected clusters on the output from any block. The output image can also be frozen in order to observe details. Typical outputs from the prototype are shown in Fig. 15 and example videos can be found on the project homepage [38].

No color features are extracted in the current version of the prototype, since the memory is not big

Figure 13. Xilinx XUP Virtex-II Pro FPGA development board with attached sensor.

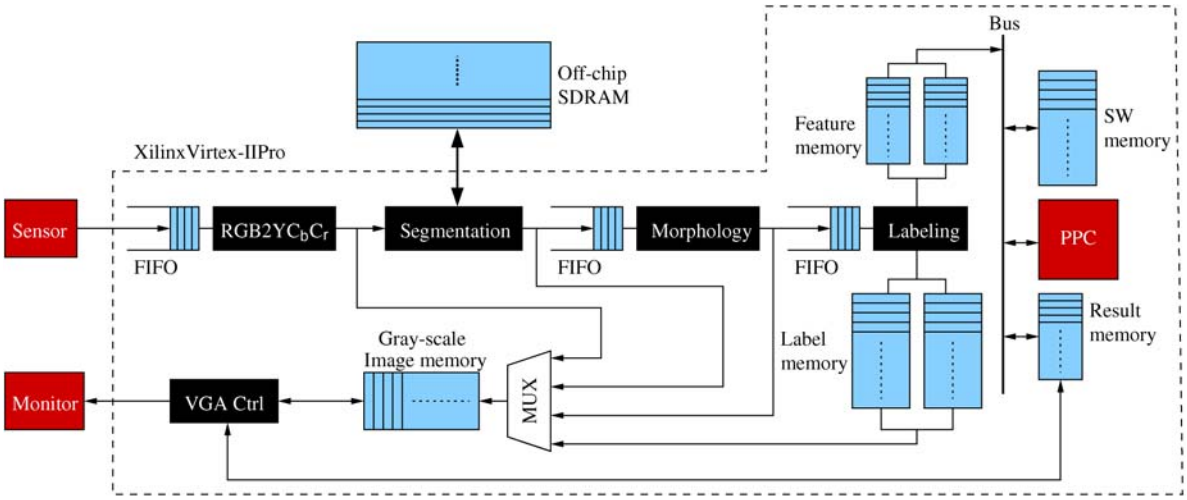

Figure 14. System architecture of the prototype, where *black* indicates custom made logic, light blue (*light grey*) is memories and red (*dark grey*) is of-the-shelf components.

enough to store a color image and the software memory is not sufficient for the complete tracking code. Current tracking software reads in all cluster features of all labeled clusters in order to draw the corresponding BB. To free on-chip memory and to be able to include color feature extraction, two additional external memories could be added to the board. One memory will contain the software and the other a complete color image.

The prototype delivers 25 fps with an image resolution of 320×240 pixels. Three Gaussian distributions per pixel, stored in an off-chip SDRAM, are used to perform color image segmentation. The morphology unit performs an opening with a flexible SE that can be of any size up to 15×15. As default, the SEs are set to 3×5 and 5×7, in the erosion and dilation block, respectively. The labeling unit extracts cluster features on up to 61 clusters per frame. The most important parameters of the different blocks are controlled with dip-switches on the board.

The chosen maximum number of labeled clusters per frame, 61, is based on SW simulations. This number together with Eq. 9 and the system environment, would with unique labels result in a total memory requirement of $\text{mem}_{tot} = FIFO + 2 \cdot (\text{mem}_{CT} + \text{mem}_{feat.}) \approx 1.06$ Mbit, where the factor 2 is due to the dual memory structure [36]. However, in our application a single label can be used without system performance degradation. Using one unique label will neither affect cluster nor color feature extraction. Cluster features are extracted during CT

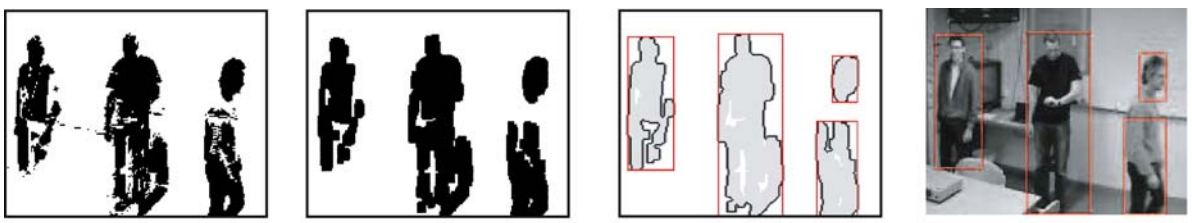

Figure 15. Typical results from the different units, **a** segmentation result, **b** the output from the morphological unit after an opening has been performed, **c** labeled output, **d** original video. The BB shown in **c** and **d** are generated from the PPC and can be applied to any of the outputs.

and color features are extracted on single objects even before the BB of another object overlap, i.e. in either case label ambiguity is avoided. The result is that the label memory only uses two bits per pixel and the complete label unit requires 438 kbit instead of 1.06 Mbit.

There are two main purposes of the prototype apart from verifying functionality; one is to perform high speed testing of different configurations, settings and long-term effects of the individual blocks. Software simulation of long-term effects can be extremely time consuming whereas "simulations" with the prototype is performed in real-time. To facilitate repeatability, the sensor is disconnected and input is read from file instead. The second purpose of the prototype is to find system bottlenecks. The required hardware resources are shown in Table 3 together with the speed of all blocks. Lookup tables (LUTs) shows how much logic that is required and the 18×18 multipliers are hard macro multipliers in the FPGA. Internal and external memory refers to the on-chip block memories and the off-chip SDRAM, respectively. The speed required to reach system performance of 25 fps is shown as operating frequency and the standalone speed of a block is shown as maximum frequency. It is seen that the morphology block is very small compared to the other parts of the system and that RGB to YC_bC_r conversion does not add significant amount of resources to the segmentation unit. The performance of the system is to a large extent dependent on the segmentation quality. Hence, the great attention on segmentation improvements such as the right color space and reduced wordlengths and memory bandwidths. Despite the improvements, measured in LUTs, multipliers, and external memory, the segmentation unit still requires most hardware. However, none of these resources are critical on a system level and will not be critical even when the color feature block is added. On a system level, internal memory is critical. Almost 75% is used and most of it is due to the grey-scale image that is stored in the VGA-controller. To extend the system to store and display color images, off-chip memory is required.

6.1. *Bottlenecks*

The presented system uses a resolution of 320×240, which is rather low compared to modern digital video cameras. This resolution is used due to the limited amount of resources, especially memory, on the FPGA. However, future surveillance systems will most likely require higher resolution. Therefore it is of interest to study system bottlenecks and how they react to an increased resolution while maintaining real-time performance. For example, if the resolution increases to 640×480, i.e. four times as many pixels per image, and the frame rate remains 25 fps. How will this affect the different parts of the system and what can be done to decrease the impact of an increased resolution?

The segmentation algorithm scales linearly, i.e. the critical memory bandwidth increases to 4.3 Gbit/s with the straight forward implementation and to 0.82 Gbit/s with the presented memory reduction scheme. To reduce the bandwidth further the approach presented in Magee [39] could be used, where the distributions are not updated every frame. The morphology unit is much less affected by a resolution increase, since the memory is only dependent on

Table 3. Hardware resources and utilization of the different parts of the prototype.

System part	RGB to YC_bC_r	Segmentation	Morphology	Label	Color feature	Track	VGA Ctrl	Total use	FPGA board
LUTs	0.8%	**12.4%**	0.6%	8.1%	14.8%	0%	1.2%	39%	27392
18×18 mult	2.2%	**5.2%**	0%	1.5%	6.6%	0%	0%	24%	136
Mem_{int} [kbit]	0%	9.6%	1.5%	20%	0%	11.8%	**32%**	**74.9%**	2176
Mem_{ext} [Mbit]	0%	**0.5%**	0%	0%	0%	0%	0%	0.5%	2048
PowerPC	0%	0%	0%	0%	0%	**50%**	0%	50%	2
$f_{operational}$ [MHz]	8	8	9	67	N.A.	100%	25	–	–
f_{max} [MHz]	N.A.	83	146	70	100	300%	N.A.	–	–

Figures for the segmentation block includes sensor control logic, and the VGA controller includes both result and image memory. Color feature extraction is currently not part of the prototype but is included for comparison.

the width of the image. If the SE is increased to match the higher resolution, i.e. to 31×31 pixels, only 2.5 times more memory is required in the data path and the intermediate FIFOs are unaffected. In the label unit, both label memories increase with a factor of 4. One way to reduce this could be to only keep one label memory used by the CT algorithm, and compress the resulting labeled image into a smaller memory using a compression scheme, e.g. run length encoding or JBIG [40]. In terms of memory, feature extraction is unaffected by the resolution increase, since it only works on streaming data and only stores the result. However, it will require 4 times as many clock cycles to execute; this is true for all previous blocks as well. The only part totally unaffected by the resolution increase is the tracking part. Neither the number of objects nor the number of features per object is affected by a resolution increase.

7. Conclusions

In this paper, an embedded automated digital surveillance system with real-time performance is presented. The system has been developed in order to identify and propose solutions to computational and memory bottlenecks. Due to the real-time processing, it also substantially reduces analysis of long terms effects due to changes in the algorithms and to parametric changes.

The main bottleneck of image processing algorithms is the high memory requirements. Therefore, a new memory scheme in video segmentation using wordlength reduction and pixel locality is proposed, reducing memory bandwidth with more than *70%*. A morphological datapath unit with a memory requirement that only scales with image width is presented. It is also shown that in our application, the labeling memory can be reduced with more than *50%* if a CT algorithm is used. On a system level, it is shown that on-chip memory is the main bottleneck. A system prototype has been implemented and is running in 25 fps on an FPGA development board.

Acknowledgments

The authors would like to thank Anders Olsson and Daniel Elvin at AXIS Communications for valuable input and support, and Xilinx for donation of FPGA development boards.

References

1. AXIS Communications, "Official AXIS Communications Homepage," 2007, http://www.axis.com.
2. Sony, "Official Sony Homepage," 2007, http://bssc.sel.sony.com/.
3. IBM, "Official IBM homepage," 2007, http://www.research.ibm.com/peoplevision/.
4. I. Haritaoglu, D. Harwood, and L. Davis, "W^4: Real-time Surveillance of People and their Activities," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, 2000, pp. 809–830.
5. C. Stauffer and W. E. L. Grimson, "Learning Patterns of Activity Using Real-time Tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, 2000, pp. 747–758.
6. R. Collins, A. Lipton, H. Fujiyoshi, and T. Kanade, "Algorithms for Cooperative Multisensor Surveillance," *Proc. I.E.E.E.*, vol. 89, no. 10, 2001, pp. 1456–1477.
7. T. Zhao and R. Nevatia, "Tracking Multiple Humans in Complex Situations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 9, 2004, pp. 1208–1222.
8. W. Hu, T. Tan, L. Wang, and S. Maybank, "A Survey on Visual Surveillance of Object Motion and Behaviors," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 34, no. 3, 2004, pp. 334–353.
9. R. Aguilar-Ponce, J. Tessier, A. Baker, C. Emmela, J. Das, J. Tecpanecatl-Xihuitl, A. Kumar, and M. Bayoumi, "VLSI Architecture for an Object Change Detector for Visual Sensors," in *Proceeding of IEEE Workshop on Signal Processing Systems Design and Implementation, SIPS*, Athens, 2005, pp. 290–295.
10. S. Fahmy, P. Cheung, and W. Luk, "Novel FPGA-based Implementation of Median and Weighted Median Filters for Image Processing," in *Field Programmable Logic and Applications, 2005. International Conference*, 2005, pp. 142–147.
11. P. Schumacher, K. Denolf, A. Chilira-RUs, R. Turney, N. Fedele, K. Vissers, and J. Bormans, "A Scalable, Multi-stream MPEG-4 Video Decoder for Conferencing and Surveillance Applications," in *Image Processing, 2005. ICIP 2005. IEEE International Conference*, vol. II, 2006, pp. 886–889.
12. R. Kordasiewicz and S. Shirani, "ASIC and FPGA Implementations of H.264 DCT and Quantization Blocks," in *Image Processing, 2005. ICIP 2005. IEEE International Conference*, vol. III, 2006, pp. 1020–1023.
13. R. Kleihorst, B. Schueler, A. Danilin, and M. Heijligers, "Smart Cameras Mote with High Performance Vision System," in *Workshop on Distributed Smart Cameras (DSC 06)*, Boulder, 2006.
14. E. Ljung, E. Simmons, and R. Kleihorst, "Distributed Vision with Multiple Uncalibrated Smart Cameras," in *Workshop on Distributed Smart Cameras (DSC 06)*, Boulder, 2006.
15. E. Ljung, E. Simmons, A. Danilin, R. Kleihorst, and B. Schueler, "802.15.4 Powered Distributed Wireless Smart Cameras Network," in *Workshop on Distributed Smart Cameras (DSC 06)*, Boulder, 2006.
16. R. Gonzalez and R. Woods, *Digital Image Processing*, 2nd ed. Prentice-Hall, 2002.
17. J. Toyama, B. Brumitt, and B. Meyers, "Wallflower: Principles and Practice of Background Maintenance," in *Proc. IEEE International Conference on Computer Vision and Pattern Recognition*, 1999.

18. C. Stauffer and W. E. L. Grimson, "Adaptive Background Mixture Models for Real-time Tracking," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, Ft. Collins, 1999.
19. G. Russo and M. Russo, "A Novel Class of Sorting Networks," *IEEE Trans. Circuits Syst. 1 Fundam. Theory Appl.*, 1996.
20. H. Jiang, H. Ardö, and V. Öwall, "Real-time Video Segmentation with VGA Resolution and Memory Bandwidth Reduction," in *Proc. of AVSS*, 2006.
21. F. Kristensen, P. Nilsson, and V. Öwall, "Background Segmentation Beyond RGB," in *Proc. of Asien Conference on Computer Vision (ACCV'06)*, Hyderabad, 2006.
22. J. Serra, *Image Analysis and Mathematical Morpohology, Vol 1*. Academic Press, 1982.
23. M. von Herk, "A Fast Algorithm for Local Minimum and Maximum Filters on Rectangular and Octagonal Kernels," *Pattern Recogn. Lett.* vol. 13, no. 7, 1992, pp. 517–521.
24. H. Hedberg, F. Kristensen, P. Nilsson, and V. Öwall, "A Low Complexity Architecture for Binary Image Erosion and Dilation Structuring Element Decomposition," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'05)*, Kobe, 2005.
25. H. Park and R. Chin, "Decomposition of Arbitrarily Shaped Morphological Structuring Elements," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, 1995, pp. 2–15.
26. G. Anelli and A. Broggi, "Decomposition of Arbitrarily Shaped Binary Morphological Structuring Elements Using Genetic Algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, 1998, pp. 217–224.
27. S. Fejes and F. Vajda, "A Data-driven Algorithm and Systolic Architecture for Image Morphology," in *Proc. of IEEE Image Processing (ICIP '94)*, Austin, 1994, pp. 550–554.
28. J. Velten and A. Kummert, "FPGA-based implementation of variable sized structuring elements for {2D} binary morphological operations," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'03)*, Bangkok, 2003, pp. 706–709.
29. A. Rosenfeld and J. Pfaltz, "Sequential Operations in Digital Picture Processing," *J. ACM*, vol. 13, no. 1, 1966, pp. 471–494.
30. M. B. Dillencourt, H. Samet, and M. Tamminen, "A General Approach to Connected-component Labeling for Aritrary Image Representations," *J. ACM*, vol. 39, no. 2, 1992, pp. 253–280.
31. W. Kesheng, O. Ekow, and S. Arie, "Optimizing Connected Components Labeling Algorithms," in *SPIE International Symposium on Medical Imaging*, San Diego, 2005.
32. K. Suzuki, H. Isao, and S. Noboru, "Linear-time Connected-component Labeling Based on Sequential Local Operations," *Comput. Vis. Image Underst.*, vol. 89, 2003, pp. 1–23.
33. F. Chang, C. J. Chen, and C. J. Lu, "A Linear-time Component-labeling Algorithm Using Contour Tracing Technique," *Comput. Vis. Image Underst.*, vol. 93, 2004, pp. 206–220.
34. PETS, "The PETS 2001 Data Set, Sequence from Camera 1," 2001, http://www.cvg.cs.rdg.ac.uk/cgi-bin/PETSMETRICS/page.cgi?dataset.
35. L. Yang and F. Algbregtsen, "Discrete Green's theorem and its application in moment computation," in *Int. Conf. on Electronics and Information Technology*, Beijing, 1994.
36. H. Hedberg, F. Kristensen, and V. Öwall, "Implementation of Labeling Algorithm Based on Contour Tracing with Feature Extraction," in *Proc. of IEEE ISCAS'07*, New Orleans, 2007.
37. E. Brookner, *Tracking and Kalman Filtering Made Easy*, 1st ed. Wiley-Interscience, 1998.
38. Lund University, "Project homepage," 2007, http://www.es.lth.se/Digital_Surveillance.
39. D. Magee, "Tracking Multiple Vehicles Using Foreground, Background and Motion Models," *Image Vis. Comput.*, vol. 22, 2004, pp. 143–155.
40. JBIG Committee, "Official JBIG homepage," 2007, http://www.jpeg.org/jbig/index.html.

Fredrik Kristensen reached the M.Sc. degree in Electrical Engineering at Lund Institute of Technology, Lund University. In May 2004, he reached the degree Licentiate of Engineering in the field of Hardware implementation of baseband OFDM at Lund University. His main research area is in real-time video processing, especially in the field of surveillance applications. He is employed by AXIS Communication but is pursuing his Ph.D. degree at the digital ASIC group at the Department of Electroscience, Lund University.

Hugo Hedberg received the M.Sc. degree in Electrical Engineering degree at Lund Institute of Technology, Lund University. He joined the digital ASIC group at the Department of Electroscience in March 2002 where he received the degree Licentiate of Engineering in the field of hardware accelerators for automated digital surveillance systems in May 2006 and is currently working towards his Ph.D. degree. His main research area is hardware implementations of image processing algorithms in real-time applications with a special interest in developing low complexity architectures for morphological operations.

Hongtu Jiang received the M.Sc. degree in Electrical Engineering from Jilin University, P. R. China in 2000. In September 2000, he joined the digital ASIC group at the Department of Electroscience, Lund University in Sweden. His main research area has been in the hardware accelerator architecture design in image processing applications. Two research projects have been developed involving design and implementations of a real-time video segmentation unit and a real-time image convolution unit. In February 2007, he received his Ph.D. degree and joined Ericsson Mobile Platform in Lund, where he is working as a Research Engineer in the area of multimedia applications.

Peter Nilsson reached the M.Sc. degree in Electrical Engineering at Lund Institute of Technology, Lund University. In May 1992, he reached the degree Licentiate of Engineering and in May 1996 he reached the degree Doctor of Philosophy in Engineering both at Lund University. After the Ph.D. degree he began as an Assistant Professor at Department of Applied Electronics (now Department of Electroscience), Lund University, Lund, Sweden. In November 1997, he became Associate Professor at the same department and in December 2003, the degree "Docent" was awarded. He was the Program Manager for Socware Research & Education, a national program for research and Master's education on System-on-Chip, 2000-05. He was also an Associate Editor for IEEE Transactions on Circuits and Systems I, 2004-05, and is a member of the VLSI Systems and Applications Technical Committee in IEEE Circuits and Systems Society. His main interest is in the field of implementation of digital circuits.

Viktor Öwall (S'90–M'95) received the M.Sc. and Ph.D. degrees in Electrical Engineering from Lund University, Lund, Sweden in 1988 and 1994, respectively. During 1995 to 1996, he joined the Electrical Engineering Department, the University of California at Los Angeles as a Visiting Researcher on a Postdoctoral grant from the Swedish Research Council for Engineering Sciences where he mainly worked in the field of multi-media simulations. Since 1996, he has been with the Department of Electroscience, Lund University, where he is currently an Associate Professor. His main research interest is in the field of digital hardware implementation, especially algorithms and architectures for wireless communication, image processing and biomedical applications. Current research projects include combining theoretical research with hardware implementation aspects in the areas of pacemakers, channel coding, video processing, and digital holography. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING from 2000–2002 and is Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.