On the Hardware/Software Design and Implementation of a High Definition Multiview Video Surveillance System

S. C. Chan, *Member, IEEE*, Shuai Zhang, Jia-Fei Wu, Hai-Jun Tan, J. Q. Ni, and Y. S. Hung

***Abstract*—This paper proposes a distributed architecture for high definition multiview video surveillance system. It adopts a modular design where single view/stereo intelligent internet protocol (IP)-based video surveillance cameras are connected to a front-end field programmable gate array (FPGA) board(s) which are connected to a back-end local video server through the IP network. The data intensive video analytics (VA) algorithms such as background modeling, connected component labeling and single view object tracking are implemented in the FPGA using an efficient fix-point based architecture. Each back-end video server is equipped with a storage and graphics processing units for supporting high-level VA and other processing algorithms such as video decompression/display, mean depth estimation and consistent labeling. A real-time prototype system was constructed to illustrate the architecture and VA algorithms involved. Satisfactory results were obtained for both publicly available data set and real surveillance video data.**

***Index Terms*—Field programmable gate array (FPGA), graphics processing unit (GPU), intelligent video surveillance (IVS), internet protocol (IP) camera, object tracking, video analytics (VA).**

I. Introduction

With low-cost and high-resolution digital cameras, large scale internet protocol (IP)-based security or surveillance systems can be conveniently connected to form distributed smart camera networks. An important application of such distributed cameras network is intelligent video surveillance (IVS) because of its importance in commercial and social security. A survey of early works in intelligent distributed surveillance systems can be found in [27]. The technological evolution of video surveillance systems started with conventional analogue CCTV systems [28], which consist of a number of cameras, usually charge-coupled device (CCD) sensors, located at different locations and connected to certain control rooms for human monitoring or recording. However, these analogue systems are limited by relatively low video resolution and the flexibility in enforcing data security, storage and retrieval, video analytics (VA) and other automated processing. On the contrary, in IP video surveillance systems [29], IP cameras can be readily connected to different computing devices, or even mobile devices, through the IP network. Different from analogue systems, digital representation and transmission over IP-based network offer improved data security and flexibility through encryption and authentication methods, which are highly desirable. Consequently, the capturing, transmission and support of VA in IVS systems [11], [20]–[22], [25], [30], [31] have received much attention recently. However, the high computational requirement of various VA tasks increases dramatically with the number and resolution of cameras. The use for high-definition (HD) videos at 720 p or 1080 p resolution will further increase the computational complexity. Therefore, most IVS systems developed are limited in the support of VA functions and resolution both locally at the cameras and distributedly over the network. Hence, it calls for more computationally efficient VA algorithms, software/hardware and system architectures to achieve real-time IVS at HD resolution.

On the architecture side, traditional IVS systems are usually developed on industrial PC platform [20], [21]. Despite the more convenient programming environment, such platforms are often criticized for its stability and reliability, especially in continuous operation under long period of time. Hardware embedded system-based architecture [22] has received much attention recently in IVS local camera architecture. For instance, the TI digital signal processors (DSPs) are used in [22] for both low-level and high-level VA tasks. Accordingly, VA tasks which were developed for software-based system need to be redeveloped in order to suit the streamlined dataflow in the DSP and possible hardware accelerators.

These motivate us to study in the paper the design and construction of an IVS to meet the emerging needs of higher resolution, VA, etc. In particular, we propose a distributed IVS system supporting HD, multiview processing, and real-time VA tasks using both embedded hardware in front-end IP cameras and distributed servers in back-end. A block diagram of the proposed system is shown in Fig. 1. It adopts a modular design where multiple front-end intelligent IP-based video surveillance cameras are connected to a back-end video server. Each server is equipped with storage and optional graphics processing units (GPUs) for supporting high-level VA and

Manuscript received February 01, 2013; accepted March 12, 2013. Date of publication May 13, 2013; date of current version June 07, 2013. This work was supported by the Innovation and Technology fund (ITF) of Innovation and Technology Commission (ITC) of Hong Kong under Project ITS/554/09FP. This paper was recommended by Guest Editor V. Tam.

S. C. Chan, S. Zhang, J.-F. Wu, and Y. S. Hung are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong (e-mail: scchan@eee.hu.hk; szjeff@eee.hku.hk; jfwu@eee.hku.hk; yshung@eee.hku.hk).

H.-Jun Tan and J. Q. Ni are with the Department of Communication and Information System, Sun Yat-sen University, Guangzhou 510275, China (e-mail: tanhaijun2007@126.com; issjqni@mail.sysu.edu.cn).

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/JETCAS.2013.2256822

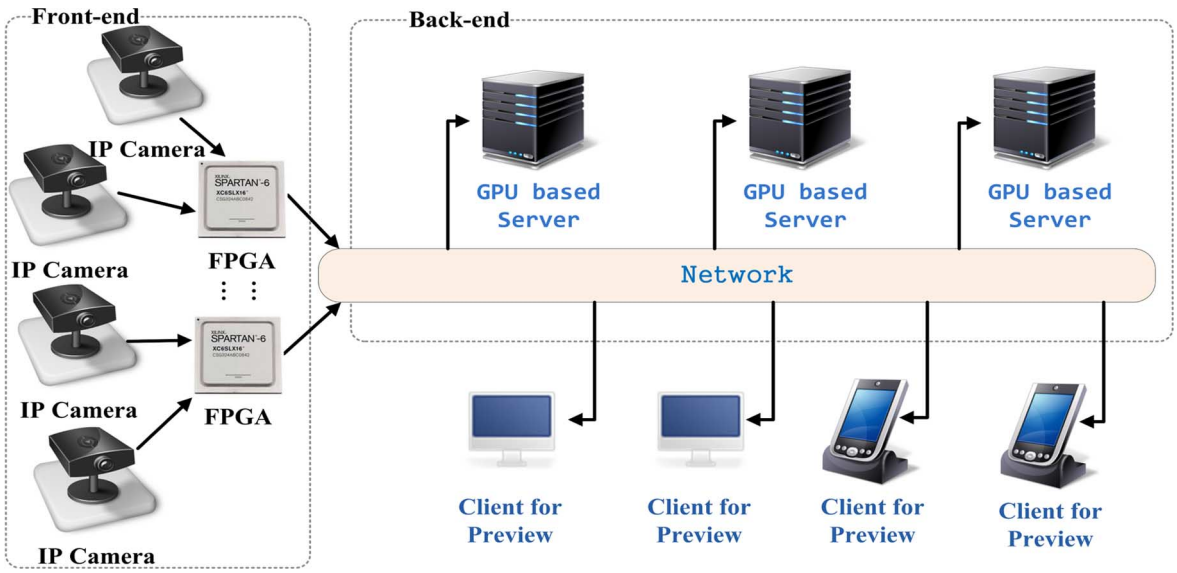


Fig. 1. Block diagram of the proposed high definition multiview intelligent video surveillance system.

processing algorithms such as real-time decoding and object tracking for the captured video. The servers are connected to the IP network for supporting distributed processing and remote data access.

The DSP-based surveillance camera is equipped with real-time algorithms for streaming compressed videos to the server. The VA tasks are performed using hardware due to its high computation requirement. The proposed system is designed for static cameras in large scale monitoring of humans and vehicles. It can be used for example in security monitoring, traffic flow measurement, accident detection on highways, speeding detection, etc. Hence, the VA tasks developed mainly focus on real-time HD multiview detection/tracking of moving objects, leading to a description about the activities of the objects in the environment.

There are mainly five major parts in our multiview tracking system, namely: background modeling (BM), connected component labeling (CCL), single view objects tracking, mean depth estimation, and consistent global objects labeling. Unlike [22], low-level VS tasks such as BM, CCL, and single view objects tracking are performed in field programmable gate arrays (FPGA), whereas high-level VAs such as mean depth estimation, multiview tracking and consistent global objects labeling are done in back-end server as it involves communications between different cameras. The graphics processing unit (GPU) servers are also equipped with more sophisticated tracking algorithm such as particle filtering [16], [17], etc., to meet different system requirements. A prototype system of the proposed IVS has been constructed. The IP cameras are based on TI-DM8127 [4], [5], which can output 720p HD videos and 1080p FHD videos at 60 frames/s (fps). The front-end VA hardware accelerator is developed in the Inravium TB-6S-LX150T-IMG2 FPGA development board while the GPU-based video server is based on an Intel i7 3.30 GHZ CPU with 8 GB RAM and 3 GTX295 GPUs. HD videos are transmitted from the IP cameras to the FPGA board through the HDMI interface. The TB-6S-LX150T-IMG2 development board contains a Xilinx Spatan-6 XC6SLX150T FPGA and three pieces of DDR3 SDRAM. It is also equipped with HDMI input as well as output. The processed results such as the position and size of the tracked objects are sent back to the IP cameras or servers through the serial port or IP network. In order to overcome the speed limitation and extensive hardware resources for supporting the stated functions in the front-end, new hardware algorithms are developed to support real time processing of HD videos. Moreover, thanks to the parallel nature of the GPU, complicated high-level VA tasks also can be realized in real-time. The VA hardware and algorithms were tested for security monitoring applications using both publicly available data set and real video data captured by our IP cameras. Satisfactory performance is obtained which substantiates the validity and usefulness of the proposed architecture and processing hardware and algorithms.

The rest of the paper is organized as follows. Section II summarizes the proposed system architecture and VA tasks, their functionalities and design challenges. The hardware design of the low-level VA tasks developed is given in Section III. Section IV is devoted to the high-level VA tasks developed for mean depth estimation, multiview objects tracking and consistent global objects labeling. Experimental results and simulations are shown in Section V. Finally, conclusion is given in Section VI.

II. Video Surveillance System Architecture and Hardware Design

As mentioned earlier, the proposed IVS consists of front-end IP cameras with hardware accelerated low-level VAs and back-end servers for distributed processing and high-level VAs. Each GPU-based server is equipped with a storage and GPU(s) for recording the compressed videos and supporting high-level VA and other processing algorithms. The video server receives compressed video and tracking results from the IP cameras (or

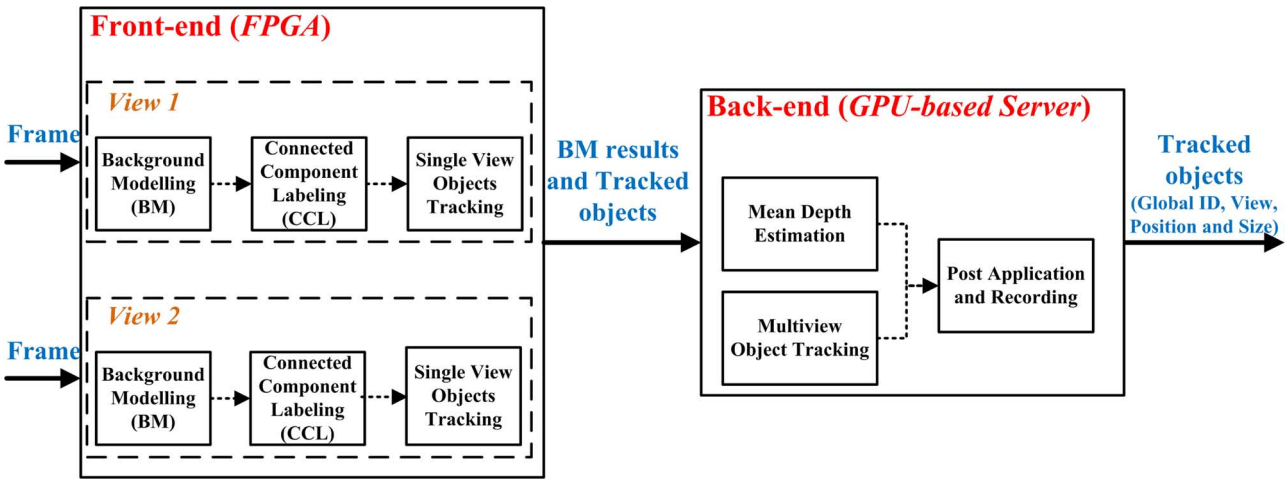


Fig. 2. Pipeline of the video analytics functions developed for the high definition multiview intelligent video surveillance system.

directly from FPGA boards for tracking results). It also decodes and displays the compressed videos for monitoring, multiview VA, and records the video data into local storage. The display and storage modules of the proposed video servers are developed using DirectShow Software Development Kit (SDK). Using DirectShow, our application can perform high-quality video and audio playback and capture. The video server can receive video data from the remote IP cameras and display the videos in different windows for monitoring.

On the other hand, the low-level data intensive VA algorithms such as BM, single view objects detection and tracking are better performed at the front-end so that immediate action can be taken. Given the limited processing power of the surveillance cameras, it is desirable to perform more complicated high-level VA algorithms mentioned at the back-end server of the distributed server network. Fig. 2 shows the block diagram of the VA pipeline, which consists of two major processing units. The front-end processing unit is located at the camera side where low-level VAs such as BM and single view object detection and tracking are implemented. On the other hand, high-level VA tasks are implemented on the back-end processing unit, which is based on a GPU-based server connected to the distributed server network. We now summarize in turn the major functionalities and design challenges of the front-end and back-end.

1) Front-End Processing Unit: This processing unit takes in new video frames captured by stereo IP cameras. It performs BM, CCL, and single view object detection and tracking on the FPGA. In our system, BM is implemented using an adaptive GM model (GMM)-based background model which performs foreground/background segmentation for each pixel captured. Traditional pixel-level GMM-based BM algorithms [13], [14] employ a fixed number of components. In the proposed system, an adaptive GMM [12], [26] is employed so that an appropriate number of components for each pixel can be automatically selected when updating the GMM parameters. This improves processing speed, since the read/write operations of the GMM parameters can be significantly reduced. After BM, if new objects are detected, an improved CCL is used to further refine the BM results. To allow the front-end to work completely on its own for alarm reporting etc., single view object detection/tracking are also implemented in the front-end.

The main design challenge of the proposed front-end processing unit is the high memory and computational requirements and limited resources in FPGA. For instance computing the probability density function (*pdf*) and updating GMM parameters in BM involve division and exponential operations, which are hardware expensive and limited in speed. Moreover, it is expensive to implement floating-point calculation in the FPGA. Hence, a carefully designed fixed-point quantization scheme has to be devised to reduce the hardware resources required, while preserving the required accuracy. In the single view object detection/tracking module, traditional object tracking method such as mean shift [15] or particle filtering [16], [17] are also hardware expensive.

To address these design challenges, a number of simplified or improved methods are proposed to reduce the hardware resources required while preserving good general purpose performance. For instance, fixed-point processing is used throughout the hardware module to reduce the resources required. Hardware efficient table look-up methods are devised to replace the division and exponential operations in the BM module, while preserving the required accuracy and achieving a higher operating speed. Furthermore, an enhanced two-pass algorithm for CCL is developed for foreground mask cleaning.

In the single view tracking module, which is based on blobs (objects) matching, a simplified hardware object tracker utilizing the velocity information is proposed to reduce hardware resource at slight expense of performance. More precisely, our method uses the GM results and velocity of moving object as input. The key advantage of our tracker is that it avoids employing color information (color histogram) and feature points which will require much hardware resources in HD resolution as compared to traditional trackers [15]–[17]. Furthermore, our tracker can also handle short term occlusions by using velocity matching.

2) Back-End Processing Unit: The purpose of the back-end processing unit is to perform VA tasks which require data from multiple cameras and possible distributed processing over the

network. It can also complement, if necessary, the limited performance of the front-end low-level VA tasks due to power and cost constraints by performing sophisticated tracking and related postprocessing algorithms with GPU acceleration.

In the proposed system, mean depth estimation and multiview consistent global object labeling of moving objects are supported. Depth information is becoming increasingly important in high-level multiview VA tasks because of its usefulness in more reliable tracking, localization and speed estimation. However, traditional dense stereo/multiview depth estimation algorithms usually require high computational complexity, especially at 720 p or 1080 p resolution. On the other hand, most IVS systems, mainly focus on the location of moving objects rather than the detailed 3-D structure. Therefore, the depth accuracy required is considerably lower than some computer vision tasks such as 3-D reconstruction and modeling. Hence, it calls for more computationally efficient depth estimation method for IVS applications. In our proposed system, the BM results are utilized to compute the mean depth of newly detected and tracked objects. By assuming that the IP cameras are calibrated, the mean depths of the tracked objects can be determined using object-based stereo matching. Other feature-based approaches can also be applied. This considerably simplifies the depth estimation process and the detail will be discussed in Section IV. Furthermore, the proposed single view tracking framework can be extended to multiview tracking across multiple cameras with the help of mean depth estimation and the details can be found in Section IV. We now discuss the design and implementation of the low-level VA algorithms developed in FPGA.

III. Design and Implementation of Low-Level Video Analytics Algorithms on FPGA

As mentioned earlier, the low-level data intensive VA algorithms are better performed at the camera side so that immediate action can be taken. Due to limited processing power of surveillance cameras, it is highly desirable to implement these algorithms in hardware, which serves as a co-processor to the main controller such as a DSP. In general, these low-level VA tasks include 1) updating the background model at each pixel, 2) blob extraction using CCL so as to detect possible foreground objects, and 3) tracking of detected objects for recording and high-level analysis.

While there are previous works on hardware design of BM [26], CCL [9], [10] and single view object tracking [6], it is usually targeted for standard definition. In this work, we propose new hardware architectures for BM, CCL, and object detection and tracking for HD videos. It differs from the previous works in [6], [9], [10], [26] in the following aspects.

1) new table lookup methods are employed to a) implement division in updating the GMM parameters to significantly overcome the speed and resource limitation of the divider intellectual property core (IP core) in FPGA, and b) to implement the exponent function required in GMM update using a piecewise look-up table method to significantly reduce the table size and hence hardware resource.
2) Fixed-point implementation: as floating-point implementation is hardware expensive, fixed-point quantization method with different fixed lengths is employed in updating the parameters of GMM components and computing the *pdf*
3) Enhanced two-pass algorithm for CCL: an enhanced two-pass algorithm based on multi-pass algorithm and two-pass algorithm is proposed. It achieves similar performance as two-pass algorithm with a lower computational complexity and simpler implementation in FPGA as multi-pass algorithm.
4) In single view tracking, several new techniques such as image frame classification in bounding box distance measure [1], [6], new storage strategy in blob-list updating, and use of velocity information in occlusion handling and tracking are proposed to reduce the hardware resource and achieve real-time operation.

We first describe the principle of BM and the proposed hardware architecture and implementation in Section III-A below. The design and hardware implementation of the blob extracting using CCL and single view objects tracking will be described in Sections III-B and III-C, respectively. The outputs of these processing modules are foreground/background masks and tracked blobs (objects) lists. In Section IV, these proposed techniques will be further extended to multiview objects tracking with mean depth estimation and consistent global objects labeling.

A. Background Modeling

1) Adaptive Gaussian Mixture Model: The main objective of pixel-based BM is to decide whether the pixel under consideration belongs to the background (BG) or foreground (FG). This can be formulated as a Bayes model selection problem by computing the Bayes factor [12] which is the ratio of the posterior probabilities of current pixel being the BG or FG. More precisely, let the pixel under consideration at time t be $x^{(t)}$. The following Bayesian factor R is used to make the inference or decision:

$$R = \frac{p\left(\mathrm{BG} \mid x^{(t)}\right)}{p\left(\mathrm{FG} \mid x^{(t)}\right)} = \frac{p\left(x^{(t)} \mid \mathrm{BG}\right) p(\mathrm{BG})}{p\left(x^{(t)} \mid \mathrm{FG}\right) p(\mathrm{FG})}. \tag{1}$$

If R is small, the pixel is detected as FG and vice versa. In general, we do not know the distribution of the FG, and hence it is common to assume that it takes a uniform distribution with $p(x^{(t)} \mid \mathrm{FG}) = c_{\mathrm{FG}}$. Consequently, if a pixel's value satisfies

$$p\left(x^{(t)} \,\middle|\, \mathrm{BG}\right) > c_{\mathrm{thr}} = R c_{\mathrm{FG}} \tag{2}$$

where c_{thr} is an appropriate threshold value to be chosen for detection, this pixel is treated as BG. The remaining task is then to update $p(x^{(t)} \mid \mathrm{BG})$, which is called the background model, from previous data χ. For clarity, we shall denoted the associated model by $p(x^{(t)} \mid \chi, \mathrm{BG})$.

In order to adapt to scene changes such as change of illumination, the data set χ should be updated continuously with a time window of T samples. Consequently, the data set at time t becomes $\chi_T = \{x^t, x^{t-1}, \ldots, x^{t-T}\}$. For each incoming sample, the training set χ_T and the corresponding density are updated. As these samples may contain both BG and FG information, the density estimated is actually $p(x^{(t)} \mid \chi_T, \mathrm{BG} + \mathrm{FG})$. Since the BG model changes slowly, the new FG samples will usually correspond to an outlying mode in the overall distribution, which

can then be separated. To model $p(x^{(t)} \mid \chi_T, \mathrm{BG}+\mathrm{FG})$, an efficient approach is based on the GMM with M components as follows:

$$p\left(x^{(t)} \mid \chi_T, \mathrm{BG}+\mathrm{FG}\right) = \sum_{m=1}^{M} \hat{\pi}_m^{(t)} N\left(x^{(t)}; \hat{\mu}_m^{(t)}, \left(\hat{\sigma}_m^{(t)}\right)^2 \boldsymbol{I}\right) \quad (3)$$

where $\hat{\mu}_m^{(t)}$ and $(\hat{\sigma}_m^{(t)})^2$ represent the estimated mean and variance for each Gaussian component, respectively. $\hat{\pi}_m^{(t)}$, $m = 1, \ldots, M$, is respectively the estimated probability of the mth component, and they sum up to one. An online recursive algorithm for updating the GMM parameters can be derived [12]

$$\hat{\pi}_m^{(t)} \leftarrow \hat{\pi}_m^{(t-1)} + \alpha\left(o_m^{(t)} - \hat{\pi}_m^{(t-1)}\right) - \alpha c_T \quad (4)$$

$$\hat{\mu}_m^{(t)} \leftarrow \hat{\mu}_m^{(t-1)} + o_m^{(t)}\left(\alpha \Big/ \hat{\pi}_m^{(t)}\right)\delta_m^{(t)} \quad (5)$$

$$\left(\hat{\sigma}_m^{(t)}\right)^2 \leftarrow \left(\hat{\sigma}_m^{(t-1)}\right)^2 + o_m^{(t)}\left(\alpha \Big/ \hat{\pi}_m^{(t)}\right) \times \left(\left(\delta_m^{(t)}\right)^T \delta_m^{(t)} - \left(\hat{\sigma}_m^{(t-1)}\right)^2\right) \quad (6)$$

where $\delta_m^{(t)} = x^{(t)} - \hat{\mu}_m^{(t)}$, c_T is the leaky or forgetting factor, and the constant α is used to control the influence of the current data and it is usually chosen as $\alpha = 1/T$. $o_m^{(t)}$ is the ownership of each component and it depends on the status of the component. In [12], the status of a component is said to be "close" if its Mahalanobis distance $D_m(x^{(t)})$ defined below is small enough, say less than three, which means that the input sample is sufficiently close to that component. Therefore, the ownership $o_m^{(t)}$ of that component is set to 1 and others are set to 0

$$D_m\left(x^{(t)}\right) = \sqrt{\left(\delta_m^{(t)}\right)^T \delta_m^{(t)} \Big/ \left(\hat{\sigma}_m^{(t-1)}\right)^2}. \quad (7)$$

If there is no "close" component in all the M components then a new component should be generated with $\hat{\pi}_{M+1}^{(t)} = \alpha$, $\hat{\mu}_{M+1}^{(t)} = x^{(t)}$, and $(\hat{\sigma}_{M+1}^{(t)})^2 = \sigma_0^2$, where σ_0^2 is an appropriate initial variance. Moreover, the maximum number of the components is finite, therefore the component with the smallest weight should be discarded accordingly. Using this algorithm, the intruding foreground objects usually appear as clusters with small weights. So the background can be roughly separated from the mixed information by retaining the major clusters for building the background model

$$p\left(x^{(t)} \mid \chi_T, \mathrm{BG}\right) = \sum_{m=1}^{B} \hat{\pi}_m^{(t)} N\left(x^{(t)}; \hat{\mu}_m^{(t)}, \left(\hat{\sigma}_m^{(t)}\right)^2 \boldsymbol{I}\right)$$

where B is obtained by arranging the weights of the components in descending order as follows:

$$B = \arg\min_B \left(\sum_{m=1}^{B} \hat{\pi}_m^{(t)} > (1 - c_f)\right). \quad (8)$$

c_f is the estimated fraction of the foreground objects [12]. If a new sample comes into an image and stays for a long enough time, it will also be considered as background object when its weight becomes larger than c_f.

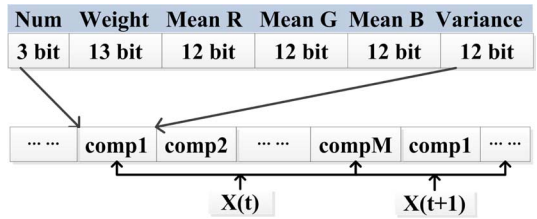


Fig. 3. Storage format of the GMM components.

2) Design and Hardware Implementation of BM: From (4)–(6), one can see that the mean, ownerships and variances of each Gaussian component need to be updated for each new incoming pixel. As this operation has to be performed for each pixel in an image frame, it calls for very high computational requirement, especially for HD videos. Fortunately, as the updating is performed for each pixel in an independent manner, it can be performed in parallel using hardware. In this work, we focus on hardware implementation using FPGA because of its reconfigurability, programmability using hardware description language (HDL) and high-level of integration. However, there are still two important problems to be addressed. Firstly, updating of the GMM parameters involves division operation, which consumes much hardware resources and is usually slow in speed. In our FPGA implementation, real-time operation cannot be achieved if the divider intellectual property core (IP core) is used even though it takes up a large number of look-up tables (LUTs). Secondly, in order to compute the background model, one needs to implement the exponential function, which is also hardware intensive. Finally, fixed-point arithmetic has to be used instead of floating-point arithmetic, as the latter is extremely hardware expensive. To address these important issues, we shall propose new table lookup methods to implement division and exponential function. A fixed-point analysis is performed to determine appropriate wordlengths in updating the various quantities in computing the GMM and *pdf*.

The storage format of the GMM parameters for each pixel is summarized in Fig. 3. It can be seen that the number of the GMM components, "Num," is stored in the entry of the first GMM parameter, "comp1." The "Num" field of the remaining GMM components of the same pixels are set as zero. Since the GMM parameters of each pixel are stored sequentially, the number of the GMM components for each pixel can be accessed conveniently.

The whole hardware architecture of the BM module is shown in Fig. 4. It mainly contains three parts: GMM updating/classification module, GMM parameters buffer and data memory. GMM updating module is in charge of computing the *pdf* of GM and updating the parameters of the GMM. GMM parameters buffer is used to realize the serial-to-parallel/parallel-to-serial conversion. Two DDR3 SDRAMs are served as data memory in our system.

As the parameters of each GMM component for each pixel are stored sequentially and the updating of each component in each pixel is done in parallel, serial-to-parallel conversion is used to retrieve the GMM parameters from the memory (the read operation) and forward them in parallel to the GMM updating/classification module at the same time. Parallel-to-se-

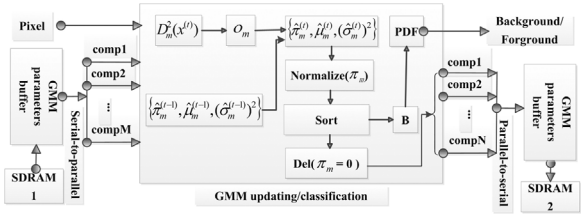


Fig. 4. Overview architecture of the background modeling.

rial conversion is also used when the parameters are written back from the module to the memory in the write operation. In order to continuously update the GMM background model, two DDR3 SDRAMs are accessed in Ping-Pong fashion so as to retrieve and store the GMM parameters. We switch the read and write operation for each DDR3 SDRAM at the rising edge of every vertical sync signal [18]. By using a variable GMM components, the bandwidth in read/write operations can be significantly reduced.

In Fig. 4, GMM parameters of one pixel enter the GMM updating/classification module concurrently. After retrieving the number of GMM components, the ownership $o_m^{(t)}$ can be obtained from the Mahalanobis distance $D_m(x^{(t)})$ in (7). If none of $o_m^{(t)}$ is "1," a new GMM component will be generated and the inputting pixel is detected as FG. Otherwise, the parameters will be updated according to (4)–(6). After the weights $\hat{\pi}_m^{(t)}$ are normalized, the B most significant clusters can be determined according to (8). The inputting pixel will be detected as BG or FG according to the threshold $c_{\rm thr}$. Before outputting the updated GMM parameters, the GMM component whose $\hat{\pi}_m^{(t)}$ is 0 will be deleted. The operation of the GMM updating/classification module for each pixel can be divided into four steps: **1**) computing the ownership for each component; **2**) updating the GMM parameters; **3**) computing the Gaussian mixture *pdf* and **4**) wordlength determination.

a) Ownership Computation: The pixel and its corresponding GMM parameters comp$i$, $i = 1, \ldots, \mathrm{M}$ are latched into the GMM classification module at the rising edge of each pixel clock simultaneously. Then the number of GMM components is extracted from the first entry, comp1. In order to decide whether a component is "close" or not, we evaluate

$$D_m^2\left(x^{(t)}\right) \cdot \left(\hat{\sigma}_m^{(t-1)}\right)^2 = \left(\delta_m^{(t)}\right)^T \delta_m^{(t)} \tag{9}$$

instead of (7), so as to avoid the hardware intensive divider. To detect whether a given component is "close" with a threshold value of 3 for $D_m(x^{(t)})$, one can compare $(\delta_m^{(t)})^T\delta_m^{(t)}$ in (9) against $9 \cdot (\hat{\sigma}_m^{(t-1)})^2$. If the former is larger, then this component is "close" and the ownership $o_m^{(t)}$ will be set to 1, otherwise it will be set to 0. So $o_m^{(t)}$ of each component will be obtained at the same time.

b) GMM Parameter Updating: From (4)–(6), we can see that the variables $\delta_m^{(t)}$, $(\delta_m^{(t)})^T\delta_m^{(t)}$ and $\hat{\sigma}_m^{(t-1)}$ are obtained during the ownership computation. These data will be reused by storing them in register blocks. The factor $\alpha/\hat{\pi}_m^{(t)}$ in both (5) and (6), however, requires a division. In order to reduce the large hardware resource and overcome the speed limitation of the divider intellectual property (IP) core in FPGA, a table look-up is employed, which will be further elaborated later in this section.

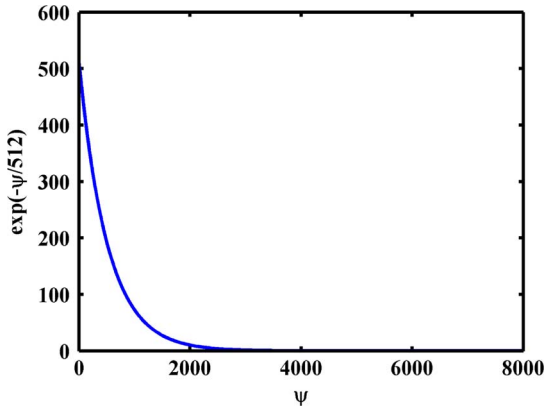


Fig. 5. Illustration of $\exp(-\Psi/512)$.

TABLE I
QUANTIZATION STEPS FOR DIFFERENT Ψ

Ψ	Quantization steps for different segments
0~511	1
512~1023	2
1024~2047	4
2048~4095	8
4096~8191	32

c) Probability Density Function Computation: In order to select the most significant B GMM components for each pixel, one needs to evaluate the *pdf* in (3), which requires the evaluation of the exponential function. There are many ways to implement an exponential function on FPGA which includes the CORDIC IP core, Taylor expansion, and Table-driven methods [19]. Considering the precision and complexity in background model, a piecewise look-up table method is adopted in this work to implement the exponent function. Since the gradient of the *pdf* $\exp(-(x^{(t)} - \hat{\mu}_m^{(t)})^T(x^{(t)} - \hat{\mu}_m^{(t)})/(2(\hat{\sigma}_m^{(t)})^2))$ decreases quite rapidly, we divide the argument into different ranges, each with a different wordlength to approximate the exponential function. To this end, we first scale up the argument $\varsigma = (x^{(t)} - \hat{\mu}_m^{(t)})^T(x^{(t)} - \hat{\mu}_m^{(t)})/(2(\hat{\sigma}_m^{(t)})^2)$ by 512 to obtain an integer for addressing the table. Therefore, $\Psi = 512 \times \varsigma$ is used as an entry to the table. We found that a 13-bits table is sufficient for our purpose and hence Ψ ranges from 0 to 8191. Thus the value of the table contains $\exp(-\Psi/512)$, which is shown in Fig. 5. In order to further reduce the table size to save hardware resource while maintaining a similar accuracy, we further divide the range of Fig. 5 into several segments. Segment with a high gradient is assigned a small quantization step while those with low gradient will be assigned a large quantization step. Hence the number of bits used to store the corresponding value of the exponential function is different at different segment/range of Ψ so as to reduce the overall storage. The quantization steps of each segment used are shown in Table I.

After GMM parameters are updated, $\hat{\pi}_m^{(t)}$ needs to be normalized as $\hat{\pi}_m^{(t)} \leftarrow \hat{\pi}_m^{(t)}/\sum \hat{\pi}_m^{(t)}$ and a similar look-up table is used

again to replace the division IP core. Afterward, all the GMM components of a pixel will be sorted in descending order of $\hat{\pi}_m^{(t)}$. According to (8), the first B largest clusters can then be identified and retained. The probability densities of the B GMM components are processed concurrently to reduce the processing time. The accumulation of the B probability densities will be compared with $c_{\rm thr}$ to determine whether the pixel under consideration belongs to the background or not (2). Before outputting the GMM parameters, the component whose weight is zero will be deleted so as to adjust the number of GMM components adaptively. To improve the performance of memory operation, the Xilinx MIG Core with a 128-bit data bus is utilized to control the memory operation [19]. In order to make full use of the Xilinx MIG Core data bus, we use a 64-bits register to store one GMM component which contains its weight, mean and variance. So we can handle 2 GMM components per read/write on the Xilinx MIG Core. As it is hardware expensive to use floating-point calculation, fixed-point computation with different fixed wordlengths is used in updating the GMM components and computing their *pdfs*.

d) Wordlength Determination: **Range of** $\hat{\pi}_m^{(t)}$: According to (4), we can see that the minimum updating step size is αc_T. If we choose the update rate as $\alpha = 0.002$ and the leaky factor as $c_T = 0.05$, then this minimum updating step size is $\alpha c_T = 0.0001$. Since the maximum value of $\hat{\pi}_m^{(t)}$ is 1, when we take $\alpha c_T = 0.0001$ as the smallest step, then the maximum steps of $\hat{\pi}_m^{(t)}$ will be "10000." In our implementation, we set $\max(\hat{\pi}_m^{(t)}) = 8191$ steps so as to fit into the nearest power-of-two number and the smallest step being $\alpha c_T = 0.0001$. Consequently, a 13-bit register with a range from 0 to 8191 will be sufficient to represent $\hat{\pi}_m^{(t)}$.

Range of $\hat{\mu}_m^{(t)}$ and related quantities: Also from (5), it can be found that the factor $\alpha/\hat{\pi}_m^{(t)}$ determines the accuracy in updating $\hat{\mu}_m^{(t)}$. The minimal value of $\alpha/\hat{\pi}_m^{(t)}$ is α when $\hat{\pi}_m^{(t)} = 1$. We found that a 9-bit register with a minimal quantization step of 0.002 gives satisfactory results. As the video input is in 24-bit RGB color format (8 bits per component), an 12-bit register is used to represent $\hat{\mu}_m^{(t)}$ whose 4-bit LSB are decimal in (5) so that the representation can be fitted into the Xilinx MIG Core data bus while achieving an appropriate accuracy. In other words, we use 36-bit to represent the various means $\hat{\mu}_m^{(t)}$ in the RGB color space.

The quantization of the variance is quite similar to the quantization of mean in (6). We found that good performance can be obtained if the variance is represented by 12 bits. The remaining 3 bits from the 64-bit register of the bus are used to indicate the number of GMM components used for the corresponding pixel. So the system can support at most 8 GMM components per pixel, which is sufficient for most applications. These 3 bits are only valid in the first GMM component while others' will be set to 0. Using this information, adaptive parallel-to-serial and serial-to-parallel conversion can be conveniently performed. Table II shows the Logic resources required for implementing the proposed adaptive GMM.

3) Connected Component Labeling (CCL): In preparation for object tracking, CCL is usually performed to detect connected regions in foreground/background binary images. Possible blobs can be obtained from the resulting binary image using a thresholding step. Several algorithms have been proposed for CCL [7]–[10]. Reference [7] presents a run-length encoding algorithm which can reduce the number of sequential operations significantly and the processing time is significantly lower than the approach in [9]. However, it may incur high memory requirement for HD videos. Unlike some other CCL algorithms that produce the labeled image, [8] proposed a single-pass streamed algorithm that extracts the data required to compute the features during the first pass. Although this algorithm does not need to produce a labeled image, its implementation on FPGA is more complex. In [9], an 8-connectivity-based multi-pass algorithm was proposed, which is quite easy to be implemented on FPGA. However, the number of passes through the image depends on the complexity of the connected components. Another two-pass algorithm, which keeps track of the pairs of equivalent labels whenever two parts of a component with a single label merge, was proposed in [10]. A bank switched structure was proposed to enable two adjacent frames to be processed simultaneously. When the first labeling pass is performed on current frame, the relabeling pass is performed on the previous frame. In this paper, we proposed an enhanced two-pass algorithm based on [9] and [10]. The proposed method has a similar performance as [10] but having a lower computational complexity and it is easy to be implemented on FPGA.

TABLE II
LOGIC RESOURCES REQUIRED IN THE IMPLEMENTATION

Resource type	Resource requirement	Percentage utilization
Slice Registers	4719	2%
Slice LUTs	2874	3%
SDP48A1s	16	8%

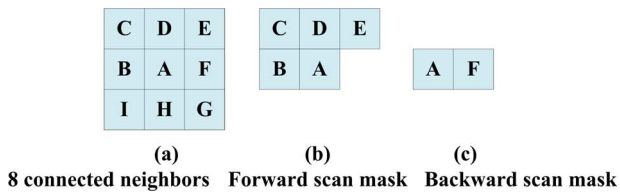


Fig. 6. Neighborhood of the current pixel A and the mask used in the first scan.

As the background image is scanned in a raster fashion, a provisional label is assigned to the object pixel in the first pass while the final label assignment is performed in the second pass (relabeling pass) according to the final equivalent table. The first labeling pass includes forward and backward scan. As an illustration, Fig. 6(a) shows the target pixel **A**, which is surrounded by eight neighbors from **B** to **I**, whereas Fig. 6(b) and (c) shows the masks used for forward and backward scans, respectively.

In the forward scan, the line containing **C**, **D** and **E** is the result of the backward scan of the last line. If the current pixel **A** belongs to the object pixel, it is assigned the minimum value among its 4 neighbor labels in the forward scan mask. Otherwise, it is assigned a value of 0. If none of its 4-connected neighbors belong to the object pixel, then the current pixel **A** is assigned a new label. An equivalent case occurs when two object

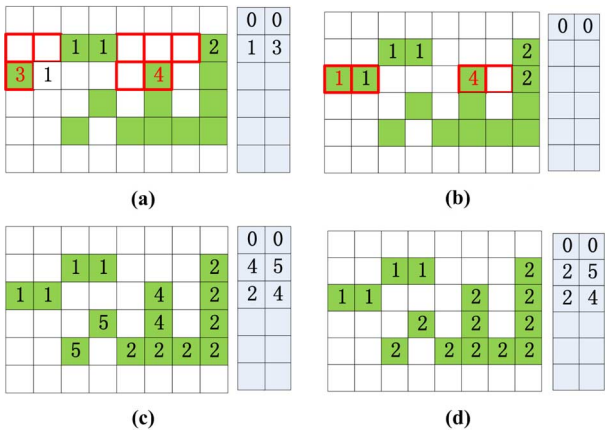


Fig. 7. An example using our proposed CCL algorithm. (a) Forward scan for the third line and the equivalent table, (b) backward scan for the third line and the equivalent table, (c) first labeling pass and the equivalent table, (d) second labeling pass and the equivalent table.

pixels with the same label are assigned two different labels in the first scanning pass. This occurs when **D** does not belong to the object while either **C** or **B**, **E** and **A** belong to the object. And the two equivalent labels are stored in a table for relabeling. In the backward scan, the current pixel is assigned the minimum label between itself and its neighbor **F**. When **A** and **F** is a pair of equivalent label, then the following procedure will be used for labeling. If none of **A** and **F** has any other equivalent relationships, then their equivalent relationship is removed from the equivalent table or else they will be reserved. In this way we can shorten the equivalent table and benefit the relabeling pass.

When compared with [9] and [10], our enhanced two-pass algorithm is beneficial when the BG image is processed by a morphology filter. More precisely, our method, unlike [9] and [10], will not generate a pair of equivalent data in the corner of the blobs, which will usually occur when morphology operation is performed on a BG image. In the relabeling pass, the object pixel is assigned the final label according to the final equivalent table, which is generated in the first labeling pass. Fig. 7 shows an example of the proposed CCL method, where green squares and white squares represent respectively the FG and BG pixels. Suppose that two new labels "3" and "4" are generated by the forward scan in the third line as in Fig. 7(a). In 7(b), by applying the backward scan mask, the pixel with label "3" is set to "1" because it is connected to its right hand side pixel with label "1." In addition, the pair of equivalent labels is deleted from the equivalent table. In this way, the length of the equivalent table can be shortened in the first labeling pass. The equivalent table and final CCL results of first labeling pass and second labeling pass are shown in Fig. 7(c) and (d).

Since the process of connected component is based on pixel, the pixel clock is used as our operation clock. In order to save processing time, the backward scan is performed following the forward scan. As for 720p videos, it costs 1650 pixel clocks to transmit each line of one image and meanwhile 1280 pixel clock are occupied by the forward scan. So there are only 370 pixel clocks remaining for the backward scan. To address this problem, we extract four adjacent pixels per pixel clock for the backward scan. By so doing, only 320 pixel clocks are required

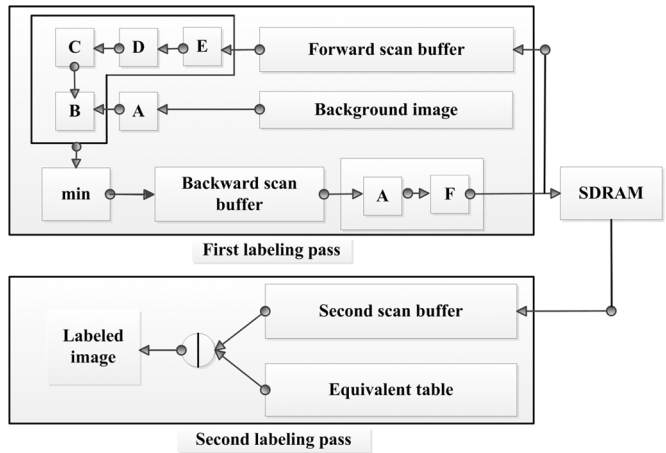


Fig. 8. Architecture of the connected component processing.

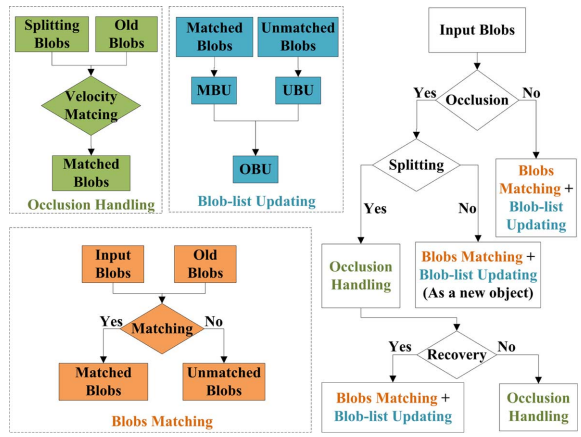


Fig. 9. Block diagram of single view objects tracking.

which allows the forward and backward scans to be performed in a pipelined fashion. The architecture of the CCL is illustrated in Fig. 8. It contains two parts which are first labeling pass and the second labeling pass. In first labeling pass, forward scan and backward scan are performed for every line in background image. The result of the first labeling pass is stored in the SDRAM. The final labeled image will be obtained by using the final equivalent table in the second labeling pass.

B. Single View Objects Tracking

As discussed earlier in Section II, single view objects tracking is performed using blobs matching. More precisely, if a new blob in a frame is matched to an old blob in the blob-list of current frame, this new blob can be considered to be successfully tracked. The object tracking contains three major operations: 1) blobs matching, 2) blob-list updating, and 3) occlusion handling as shown in Fig. 9.

It can be seen from Fig. 9 that the input blobs are first passed through the occlusion detection. If it is not occluded, blobs matching and blob-list updating, to be described in the sequel, will be used for matching and tracking, otherwise, occlusion handling will be activated to handle the occlusion. In occlusion handling, we mainly deal with the key problem of how to track the objects correctly when objects split again after occlusion. In

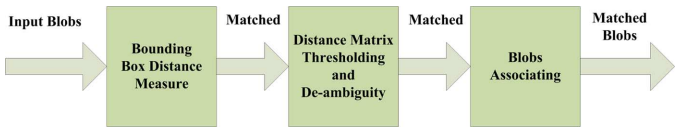


Fig. 10. Block diagram of blob matching based on bounding box distance.

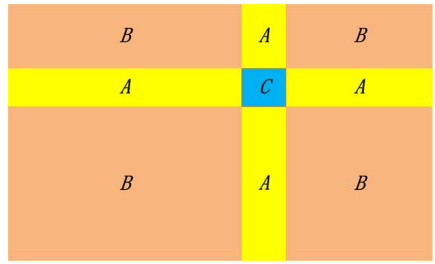


Fig. 11. Image frame classification. Blue area shows the location of old blob t_j. Areas $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are classified according to bounding box distance $D_{\mathrm{mtx}}(b_i, t_j)$.

blobs matching, it is used to judge whether the input blobs can be matched with the old blobs stored in blob-list. After blob matching, the blob-list needed to be updated in the blob-list updating module. There are three types of blobs in the blob-list, which include old blobs, matched blobs, and unmatched blobs. They are stored at their respective space or unit. For instance, the old blobs in previous frame are stored in the old blobs unit (OBU), whereas, the matched blobs unit (MBU) and unmatched blobs unit (UBU) are used to store the matched and unmatched blobs in current frame, respectively. As new blobs may arise while old or unmatched blobs may disappear, they needs to be continuously updated.

The complete hardware implementation of the proposed tracker poses several design challenges. Some of them and the proposed solution are summarized below.

1) In blobs matching, image frame classification is employed to reduce the computational complexity involved in computing the bounding box distance measure.
2) It is common to use a fixed value to perform the distance matrix thresholding. However, the performance is unsatisfactory when the scene is complicated. A velocity assisted distance matrix thresholding is therefore developed for improving the overall performance.
3) While it is common to employ dynamic storage [3] to update the bloblist in software implementation of the tracker, implementing dynamic storage at HDL or hardware is rather difficult. A new "Multiple Single-Step Updating" (MSSU) method, to be described later in this section, is therefore proposed for blob-list updating.
4) Though particle filtering and mean shift are frequently used for object tracking and handling occlusion, their hardware implementation is demanding. A simplified hardware object tracker utilizing the velocity information is therefore proposed to reduce hardware resource at slight performance degradation. We now discuss these operations in turn.

1) Blobs Matching: Blobs matching plays an important role in single view objects tracking and its implementation block diagram is summarized in Fig. 10. We can see that it contains three major operations, i.e., bounding box distance measure calculation [1], distance matrix thresholding and de-ambiguity, and blobs association. In our application, an image frame classification approach is adopted to reduce the complexity of computing the bounding box distance measure. Moreover, the velocity of the input blob is utilized to estimate the threshold in distance matrix thresholding, which can considerably enhance its performance. The three operations of blobs matching are discussed as follows.

a) Bounding Box Distance Measure: The blob can be described by a bounding box which indicates its profile and some other parameters such as size and velocity. Usually, the distance between two bounding boxes is measured by the Euclidean distance between their centroids. However, a key drawback of this measure is that a large jump may appear when two moving objects merge or split. On the other hand, if the bounding box distance between two blobs, say A and B, is measured, by the distance between the centroid of A and the closest point on B, then even if the merging or splitting happens, the variation of the bounding box distance will not be large. Thus, the undesirable large jump effect can be eliminated by using this property [1].

For each input image frame, the process attempts to associate each input blob with one of the old blobs in the OBU. This is accomplished by first constructing a distance matrix indicating the distance between each of the input blobs and old blobs. The rows of the matrix are the old blobs and the columns are the input blobs. The elements in this matrix are the bounding box distances between them. It can be seen from Fig. 11 that an image frame can be classified into three areas. The blue area $\boldsymbol{C}$ indicates the location of an old blob and the new input blob will appear either in the other two areas, $\boldsymbol{A}$ and $\boldsymbol{B}$. Conventionally, square root operation is used to calculate the bounding distance. However, if the new input blob is in area $\boldsymbol{A}$, then the square root operations can be simplified to the absolute value of their horizontal or vertical ordinates, which can reduce considerably the computational complexity. Therefore, the following simplified method is proposed for calculating the bounding box distance:

$$D_{\mathrm{mtx}}(b_i, t_j) = \begin{cases} \min \begin{pmatrix} |x_{b_i} - x_{l_j}|, |x_{b_i} - x_{r_j}| \\ |y_{b_i} - y_{p_j}|, |y_{b_i} - y_{g_j}| \end{pmatrix}, & b_i \in \boldsymbol{A} \\ \min \left\{ \begin{array}{l} \sqrt{\left(x_{b_i} - x_{l_j}\right)^2 + \left(y_{b_i} - y_{p_j}\right)^2} \\ \sqrt{\left(x_{b_i} - x_{r_j}\right)^2 + \left(y_{b_i} - y_{p_j}\right)^2} \\ \sqrt{\left(x_{b_i} - x_{l_j}\right)^2 + \left(y_{b_i} - y_{g_j}\right)^2} \\ \sqrt{\left(x_{b_i} - x_{r_j}\right)^2 + \left(y_{b_i} - y_{g_j}\right)^2} \end{array} \right\}, & b_i \in \boldsymbol{B} \\ 0, & b_i \in \boldsymbol{C} \end{cases} \tag{10}$$

where b_i and t_j are the ith input blob and jth old blob with centroids located at (x_{b_i}, y_{b_i}) and (x_{t_j}, y_{t_j}), respectively. (x_{l_j}, y_{p_j}), (x_{l_j}, y_{g_j}), (x_{r_j}, y_{p_j}), and (x_{r_j}, y_{g_j}) are the coordinates of top-left, bottom-left, top-right, and bottom-right junctions of the jth old blobs $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are classified areas of the image frame as illustrated in Fig. 11.

b) Distance Matrix Thresholding and De-Ambiguity: Once the bounding box distance has been computed, an appropriate threshold is used to obtain a binary matching matrix, which associates an input blob with an old blob. Note that the threshold is crucial since it determines the accuracy of the association. If it is too small, some existing objects may be unable to match with their new positions in the input blobs and hence be dropped. On the contrary, ambiguities may arise if it is set too large.

In this work, an improved threshold selection method is proposed so that the matching errors can be remarkably reduced. More precisely, the velocity of the input blob is utilized to estimate the threshold. One way to estimate the velocity of the input blob is Kalman filtering (KF) [2]. However, the implementation of KF in hardware is rather complicated given that many other tasks have to be implemented on the same FPGA board. To compute the velocity of the blobs, the displacement between the centroids of two matched blobs during successive frames is needed. As the old and input blobs have not yet been matched in current image frame, the blob's velocity is estimated from its values in the previous n frames as

$$\hat{v}_o^{(t)}(i) = \frac{1}{n}\sum_{j=1}^{n} \hat{v}_0^{(t-j)}(i) \tag{11}$$

where $\hat{v}_o^{(t)}(i)$ is the estimated velocity of the ith object in the image frame at time t. Once the estimated velocity is available, the displacements of each blob during successive image frame can be estimated, which can be used to derive the binarization threshold for the ith object at time t as

$$T_r^{(t)}(i) = \hat{v}_o^{(t)}(i)/F \tag{12}$$

where F is the frame rate. Once all thresholds are available, the following operations can be implemented to obtain a binary matching matrix:

$$D_b(i,j) = \begin{cases} 1, & \text{if } \left(D_{mtx}(i) < (1+\alpha)T_r^{(t)}(i)\right) \\ 0, & \text{otherwise} \end{cases}$$

where $D_b(i,j)$ is the binary value in the binary matching matrix, and i, j are indexes of the row and column, respectively. α is a forgetting factor for the threshold variation. Consequently, we obtain a corresponding matrix containing more than one nonzero elements in each row or column. Every nonzero element in the matrix indicates that one input blob and one old blob has been matched, and vice versa.

It may happen that more than one old blobs may correspond to the same input blob. To tackle this problem, we propose to use information such as the size of the blobs in addition to velocity to resolve the ambiguity. This is motivated by the fact that the size of the blob usually does not change significantly for successive image frames. In general, the size of the input and old blobs can be obtained from the coordinates of the bounding box as

$$s(w) = |x_l - x_r|, \quad s(h) = |y_p - y_g| \tag{13}$$

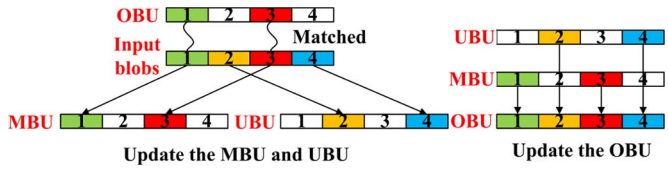


Fig. 12. Operation diagram of updating the members of blob-list.

where $s(w)$ and $s(h)$ represent the width and height of the blobs, respectively. (x_l, x_r) and (y_p, y_g) are the coordinates of the bounding box along the x-axis and y-axis. When an ambiguity occurs, the old blob with a size closest to the input blob is chosen as the true object. The corresponding matrix is updated accordingly.

c) Blobs Association: The objective of this last operation in blobs matching is to associate the matched blobs in current frame to the tracked blobs in previous frames. First we generate an unique identity number (ID) for each matched blob in MBU in the current frame. The matched blob's ID is a 8-bit number used to distinguish the matched blobs. Next, a register-table is employed to store the IDs. Finally, they are packed together with the synchronizing signals, i.e., data enable and vertical synchronization. The IDs will be used later for labeling matched objects in display.

2) Blob-List Updating: Blob-list updating mainly contains three steps, i.e., MBU updating, UBU updating and OBU updating. More precisely, we need to replace the old matched blobs in the MBU by the new matched blobs and update the UBU by new unmatched blobs. Then, the OBU is updated by combining the MBU and UBU into one unit. It is known that one of the most convenient and efficient approaches to implement blob-list updating is dynamic storage [3], where the matched and unmatched blobs are put into appropriate addresses using pointers or variable arrays in high level programming languages, such as C++ and Java. An simple procedure of blob-list updating are shown in Fig. 12. We assume that the input blobs, blob 1 and blob 3, are matched according to the OBU. Then we begin to update the blob-list. First, blob 1 and blob 3 are written into MBU. Then, the UBU is updated by unmatched blobs, blob 2 and blob 4. Lastly, blobs in MBU and UBU are combined to update the OBU. However, in Verilog HDL, it is very difficult to use dynamic storage to update the blob-list because no pointers and variable arrays can be used. For instance, one needs to preserve the matched blobs in the OBU, while replacing the old unmatched blobs by new unmatched blobs in the UBU. To handle this problem, we propose a "Multiple Single-Step Updating" (MSSU) method to update the blob-list without dynamic storage.

Briefly, the proposed MSSU involves three major steps. The first and third steps are the same as in blobs-list updating mentioned above. In the second step, instead of writing all unmatched blobs into UBU, only the first unmatched blob is put in the corresponding address of UBU. Followed by the operation of MSSU, the recursive operation is executed in the next frame until all of the new unmatched blobs are written into UBU. It should be noted that though much time is required to complete the whole updating progress due to the quantity

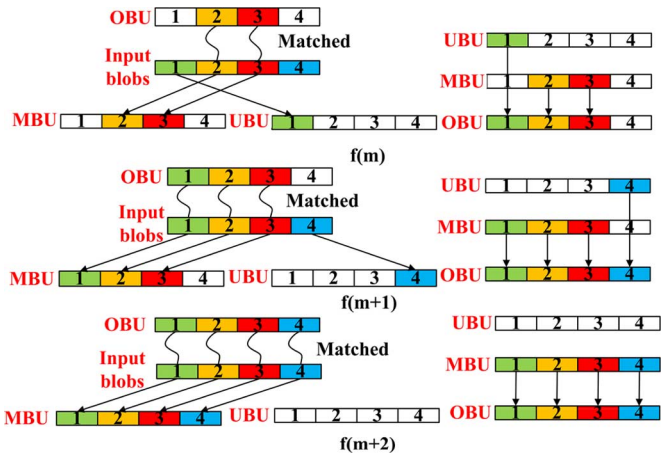


Fig. 13. Example of MSSU process.

of unmatched blobs, the influence of MSSU to the system performance is negligible for high enough frame rate, which is up to 30 fps or above. For illustration, an example operation of the proposed MSSU is shown in Fig. 13. It can be seen that there are two matched blobs, i.e., blob 2 and blob 3, in the mth frame, both of which are pushed into MBU after being matched with the blobs in OBU. On the other hand, in the unmatched blobs, i.e., blob 1 and blob 4, only the former one is written into the UBU. Then, the MBU and UBU are combined to generate the new OBU in the mth frame, which is treated as the referenced unit in the next frame. The above operation is repeated in the following frame to find out the new unmatched blob. Moreover, it can be observed from this example that only two MSSU operations are required to update the blob-list since there are two new unmatched blobs.

3) Occlusion Handling: It is known that occlusion between different objects may affect considerably objects tracking. Particle filtering [16], [17], and mean shift [15] have been proposed to address short term occlusion problems. However, the implementation of particle filtering or mean shift requires considerably hardware resources. Here, we propose a simpler alternative, aiming at a lower hardware requirement. More precisely, overlapping objects are treated as new objects when occlusion occurs. Thus, the problem reduces to how to track correctly the objects when objects split again after occlusion. For instance, we need to ensure consistency of the blob's IDs before and after occlusion, which we referred to as "occlusion recovery." During occlusion recovery, the corresponding blobs after splitting are searched in the old matched blobs. However, the position of each blob may change considerably, which makes the bounding box distance measure unreliable. Hence, the key problem during occlusion recovery is to find out the correspondence between the blobs after the splitting. Fortunately, it is observed in outdoor scene with reasonable chosen camera location that most of the occlusions does not last for a very long time. So the motion direction and velocity value are highly correlated in successive image frames, especially at high frame rates. Therefore, it is proposed to use the velocity information in matching the corresponding objects during occlusion recovery. After the recovery, the matching mode turns back to bounding box measure

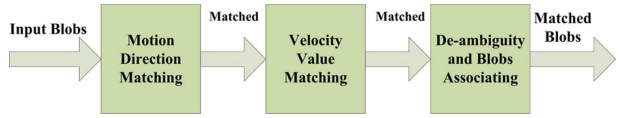


Fig. 14. Velocity matching mechanism.

as shown in Fig. 9. Otherwise, the velocity matching will continue to execute until the end of recovery. The block diagram of velocity matching mechanism is shown in Fig. 14. It can be seen that there are three operations, i.e., motion direction matching (MDM), velocity value matching (VVM) and de-ambiguity and blobs associating. In MDM, the input blobs are considered as matched only when they have the same direction with the old blobs. In VVM, a feasible variation range is used for matching.

More precisely, if the velocity of an input blob v_b and an old one v_r satisfy $v_b \in ((1-\lambda)v_r, (1+\lambda)v_r)$, where λ is user defined normalized variation factor and is set to 0.125 in our paper, then the blobs can be considered as matched, and vice versa. Once both of the above velocity parameters of the input blobs are matched, these blobs can be deemed to be matched. It should be mentioned that since multiple matched blobs may be obtained, one can employ the function of de-ambiguity to determine the best matched object and the implementation of blobs association is similar to the operation in blobs matching.

IV. High-Level Video Analytics (VA) Algorithms With Distributed GPU-Based Servers

As mentioned, the low-level VA algorithms are better performed at the camera side so that immediate action can be taken. It is desirable to perform more complicated high-level VA algorithms, such as global object labeling and tracking, in a distributed server network. This is because the IP cameras in a certain region can transmit the processed information, such as the compressed videos and blob size and location to its associated server for storage. On the other hand, the server can fuse such blob information, probably with the help of other servers, to perform global labeling and tracking of selected object or target. With high-speed internet connections, these servers can freely exchange information to perform even more sophisticated global processing in a distributed manner.

In this section, we shall illustrate this concept through two important intermediate VA tasks, namely: 1) mean depth estimation of tracked object, and 2) multiview object tracking where an object is global tracked over multiple cameras by assigning to it a global label. To our best knowledge, few IVS systems compute the depth map of objects due to the high arithmetic complexity and relatively low reliability. In our work, we only focus on the mean depth value, which is more reliable to compute and requires less computational resources. Moreover, we shall propose a new multiview tracking algorithm, which incorporates the mean depth information to provide a more reliable handoff between cameras. By using GPU acceleration, both of them can be implemented in real-time and they only consumes little system resources.

1) Mean Depth Estimation: As reliability is increasing important in IVS system, the inclusion of depth information is becoming more critical to high-level VAs. However, conventional

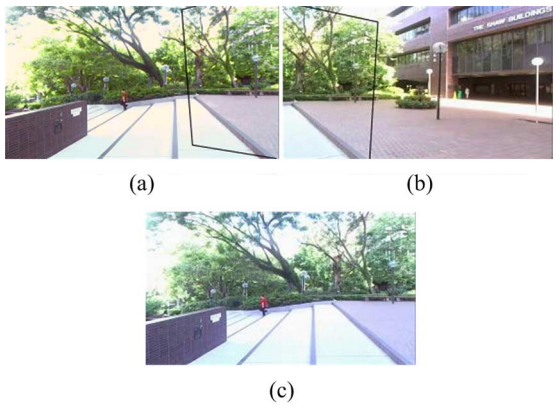


Fig. 15. Illustration of FOV lines and brightness calibration. Black lines in (a) and (b) show the FOV lines in adjacent views, respectively. Image (c) is the original view of (a) before brightness calibration.

depth estimation is time consuming and its reliability is somewhat limited. Therefore, we shall focus on mean depth of objects. To reduce the complexity in finding their mean depths, we utilize the results of BM and focus only on those detected region. We then employ stereo matching with GPU acceleration to compute depth values inside the detected object so as to estimate its mean depth. More precisely, the foreground mask detected is used to extract the texture of the object from the stereo captured by the IP cameras. Thanks to the reduced data size, a GPU accelerated stereo matching can be implemented readily to achieve real-time implementation. In our implementation, we employ conventional block matching method [23] to obtain dense depth values of the moving object. Other feature point-based method may also be applicable. The computed depth values will include depth values arising from the object as well as those coming from the background. To reliably estimate the mean depth of the foreground, the median value, and possibly other robust estimator, is used.

2) Multiview Objects Tracking: We now describe how multiview tracking can be performed in our IVS framework. An important advantage of our system is that the single view tracking results described in Section III can be readily extended to the tracking of multiple objects across multiple cameras. More precisely, the process can be divided into three steps. 1) Determination of the field-of-view (FOV) lines of each camera. 2) Brightness calibration of neighboring cameras, and 3) Consistent labeling across cameras when a new object is detected in one or more views. Since the cameras are assumed to be calibrated, the FOV lines of each view can be easily defined. We assume that the surveillance IP cameras involved in such tracking are static, since pan–tilt–zoom (PTZ) surveillance usually involves active vision and the strategies can be very different from our situations. Therefore, steps 1) and 2) in our case can be performed at the beginning of the monitoring and only an additional step of consistent labeling is required in online applications. Fig. 15 shows an example of multiview video surveillance system where FOV lines of each view are drawn in black with brightness calibration performed. The consistent (global) objects labeling method proposed in our system is motivated by the prior work in [24]. Unlike 3-D reconstruction schemes which project the location of each object in the world coordinate system, this method uses color dissimilarity and distance of object to FOV line to obtain consistent handoff of tracking-object labels across cameras. Therefore, it is efficient and the computational complexity is very low. However, if two objects have similar color distribution and distances to FOV lines are encountered, its performance will significantly degraded. Fortunately, in our system, mean depth of moving object is estimated online and hence it can further improve consistent labeling to obtain more reliable handoff, which will be described next.

3) Depth-Assisted Consistent Labeling: When the nth object O_i^n enters view V_i $i = 1, \ldots, N$, where N is the number of views, the visibility of each view of O_i^n is first checked. If O_i^n is only visible in V_i, then a new global label is assigned to O_i^n. Otherwise, O_i^n should be visible in other cameras defined in their own blob list B_{V_i}. Then the corresponding global label in the global blob list B_G is searched in the following steps.

a) A List of Corresponding Candidates in Other Stereo Views is Generated for Each Object (O_i^n) as Follows: For each stereo view V_j $j \in N$ and $j \neq i$, we search the object list for objects O_j^m which are very similar to O_i^n with $d(O_j^m, L_j^i) < \ell$ that moves toward the visible region of V_i from V_j, where $d(O_j^m, L_j^i)$ is the minimum distance between the object position of O_j^m and the FOV line L_j^i. ℓ is a tolerance threshold which can be set to an appropriate small constant value [24].

b) Computation of Dissimilarity Measures: By assuming that the distance to the FOV line is Gaussian distributed, the likelihood of matching two objects in terms of the distance to FOV line can be written as

$$P_d\left(O_i^n, O_j^m\right) = \frac{1}{\sqrt{2\pi}\sigma_d} e^{-d^2(O_j^m, L_j^i)/2\sigma_d^2} \tag{14}$$

where σ_d controls the scale or width of the region for selecting possible candidate matches. In [24], the likelihood of color dissimilarity of two objects is similarly defined as

$$P_c\left(O_i^n, O_j^m\right) = \frac{1}{\sqrt{2\pi}\sigma_c} e^{-d^2(h_i, h_j)/2\sigma_c^2} \tag{15}$$

where h_i and h_j are the color distributions (histograms) of O_i^n and O_j^m respectively. $d(h_1, h_2)$ is the Bhattacharyya distance between two distributions.

To further reduce the ambiguity of handoff between adjacent views due to errors in estimating object position, FOV lines and similarity distribution between two objects, we introduce the depth dissimilarity of two objects as follows:

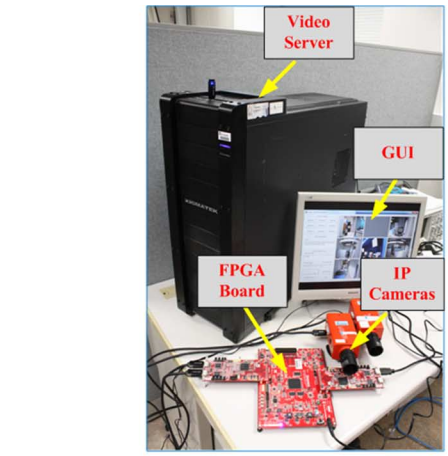
$$P_{d'}\left(O_i^n, O_j^m\right) = \frac{1}{\sqrt{2\pi}\sigma_{d'}} e^{-d'^2(O_i^n, O_j^m)/2\sigma_{d'}^2} \tag{16}$$

where $d'(o_i^n, o_j^m)$ measures the mean depth difference between O_i^n and O_j^m in view j, which can be computed from the procedure described in Section IV-A.

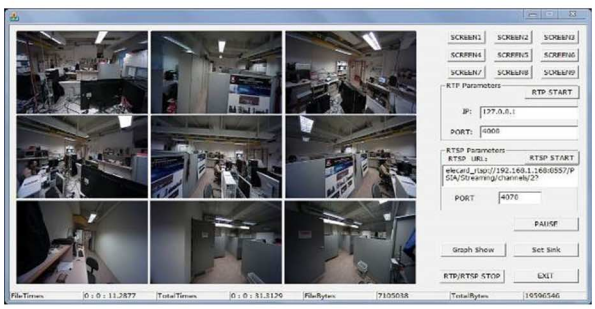
c) Assignment of Label: For each candidate, the likelihood of assigning the global label of O_i^n and O_j^m is defined as the product of P_d, P_c, and $P_{d'}$

$$P\left(O_i^n, O_j^m\right) = P_d\left(O_i^n, O_j^m\right) P_c\left(O_i^n, O_j^m\right) P_{d'}\left(O_i^n, O_j^m\right). \tag{17}$$

If the object with the highest likelihood as computed in (17) has been globally labeled in the global blob list B_G, then we assign



(a)



(b)

Fig. 16. (a) Proposed system IVS system. (b) Graphic user interface (GUI) of the software developed for the real-time monitoring among different video streams from remote IP cameras.

its global label to O_i^n. Otherwise, a new global label is assigned to both of them.

V. Experimental Results

In order to evaluate the hardware system design and VA algorithms, extensive experiments have been carried out. The VA algorithms are tested on public datasets PETS2001 [32] and real data that are captured by the IP cameras. TI-DM8127 IP cameras with 60 fps and are used for capturing. The resolution of the PETS2001 data set is 768×576 and those of the IP cameras is 720 p/1080 p for DM8127. From the results, we find that the system can steadily capture, stream, display and store HD videos. Fig. 16(a) shows the constructed prototype of the proposed IVS based on the proposed architecture in Fig. 1. It shows the GPU-based video server, IP cameras, FPGA board and the graphic user interface (GUI) of the software developed. Fig. 16(b) shows an enlarged view of the GUI, which can receive and display nine different 720 p/1080 p video streams simultaneously. The GUI can also individually control each specific video stream of the IP cameras, such as start, stop and pause. The video server supports two formats for recording the video data into files: 1) raw data format; and 2) compressed video streams. A proposed IVS GUI demo can be found at: http://youtu.be/SI3eeMUbljk. Furthermore, VA algorithms can successfully run at 20–25 frame/s on the GPU-based video servers. Here, we compare our results with color-based particle filtering [16] on PETS2001 dataset. Color-based particle

(a) (b) (c)

Fig. 17. Single view objects tracking with occlusion handling on PETS2001 dataset. The labeled rectangles indicate different tracked objects. Blue rectangles show the particle filtering tracking result and red rectangles show our single view tracking result.

TABLE III
Comparison of Maximum Frame Rate on FPGA and GPU-Based Server

	RESOLUTION	FRAME RATE
FPGA	720p	60fps
	1080p	30fps
GPU-based server	720p	25fps
	1080p	17fps

TABLE IV
Time and Resource Usage

	TIME USAGE	RESOURCE USAGE	
		LUTs	REGs
BM	16.67ms	5066	7781
CCL	33.33ms	10937	1534
Single view tracking	19.73ns	2676	2634
Others	*Na*	4728	7659
TOTAL	33.3ms	23407	19608

filtering algorithms have received great attention recently because of its ability to handle clutter and occlusions. They use color distribution (histogram) as object representation, and the procedure can be broadly divided into four parts named observation, propagation, estimation and random sampling. However, as mentioned before, the computational complexity of particle filtering methods are very high and are generally hardware expensive. Fig. 17 shows the single view objects tracking result of the PETS2001 data set. The results of the proposed tracking method and particle filtering are shown in red rectangles and blue rectangles, respectively. It can be seen that both methods can successfully track the moving objects in this data set. Because of the use of color information in [16], particle filtering can still precisely assign correct labels to different occluded objects during occlusion. In our single view tracking algorithm, the occluded objects are merged as a new object [Fig. 17(b)] and are split after occlusion [Fig. 17(c)]. The occlusion is also successfully handled since blob IDs of different objects remain valid after occlusion split.

The maximum processing frame rates of the BM, CCL and single view objects tracking with FPGA and GPU-based server implementations are shown in Table III. We can see that the processing speed of the FPGA significantly outperforms the GPU-based server both in 720 p and 1080 p resolution. The time and resource usages of the FPGA implementation in 1080 p resolution are described in Table IV. "*Na*" in the table means the time usage can be ignored. It should be noted that the total time usage of the hardware implementation does not equal to the sum of all modules because BM, CCL and single view objects tracking, are executed in parallel.

Fig. 18. Two-view outdoor tracking results corresponding to frame number 208, which includes two adjacent views. The red rectangle(s) in the upper figures indicate the tracked object(s), while the lower figures depict the corresponding background subtraction results and mean depth estimation results. Note that the object which appeared in both views is successfully identified as the same object because of consistent labeling.

Fig. 18 illustrates the outdoor multiview objects tracking results. In the upper figures of Fig. 18, depth-assisted consistent global labeling is performed when objects cross the FOV lines of adjacent cameras. The labels of the tracked objects are successfully maintained when they pass through the FOV lines from one camera to another. In the lower figures, the corresponding background subtraction results and mean depth estimation results are depicted. More tracking results can be found in our demonstration video at: http://www.youtube.com/watch?v=bbX7UMXPvL8.

VI. Conclusion

The design and implementation of a HD multiview IVS system has been presented. It adopts a modular design where multiple intelligent IP-based surveillance cameras are connected to FPGA front-end and GPU-based back-end video server. The data intensive VA tasks are performed in FPGA, while further processing such as mean depth estimation and multiview object tracking are performed at back-end. New hardware efficient and GUP-based algorithms for the front- and back-ends are also presented. A real-time prototype system was constructed to illustrate the architecture and VA algorithms involved.

References

[1] A. Senior, A. Hampapur, Y. L. Tian, L. Brown, S. Pankanti, and R. Bolle, "Appearance models for occlusion handling," *J. Image Vis. Comput.*, vol. 24, no. 11, pp. 1233–1243, 2006.

[2] S. C. Chan, B. Liao, and K. Tsui, "Bayesian Kalman filtering, regularization and compressed sampling," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Aug. 2011, pp. 1–4.

[3] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," *J. Memory Manage.*, pp. 1–116, 1995.

[4] IP Camera TI-DM8127 [Online]. Available: http://www.ti.com/apps/docs/mrktgenpage.tsp?contentId=41246&appId=79&DCMP=dsp\textunderscore videosecurity&HQS=ipcamera&247SEM

[5] IP Camera TI-DM8127 [Online]. Available: http://www.Appropho.com/NewWeb/Product_DM8127J3.php

[6] Y. Wang, J. F. Doherty, and R. E. Van Dyck, "Moving object tracking in video," in *Proc. IEEE Workshop Appl. Imagery Pattern Recognit.*, 2000, pp. 95–101.

[7] K. Appiah, A. Hunter, P. Dickinson, and J. Owens, "A run-length based connected component algorithm for FPGA implementation," in *Proc. Int. Conf. Elect. Commu. Eng. Tech.*, Dec. 2008, pp. 177–184.

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

[9] D. Crookes and K. Benkrid, "An FPGA implementation of image component labeling," in *Proc. SPIE Reconfigurable Technol.: FPGAs Comput. Appl.*, Sep. 1999, pp. 17–23.

[10] R. V. Rachakonda, P. M. Athanas, and A. L. Abbott, "High-speed region detection and labeling using an FPGA-based custom computing platform," in *Proc. Int. Workshop Field Program. Logic Appl.*, Sep. 1995, pp. 86–93.

[11] S. Zhang, S. C. Chan, R. D. Qiu, K. T. Ng, Y. S. Hung, and W. Lu, "On the design and implementation of a high definition multi-view intelligent video surveillance system," in *Proc. IEEE Int. Conf. Signal Process. Commu. Comp.*, Aug. 2012, pp. 353–357.

[12] Z. Zivkovic and F. Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern Recognit. Lett.*, vol. 27, pp. 773–780, May 2006.

[13] P. W. Power and J. A. Schoonees, "Understanding background mixture models for foreground segmentation," in *Proc. Int. Conf. Image Vis. Compt.*, Nov. 2002, pp. 267–271.

[14] M. Harville, "A framework for high-level feedback to adaptive, per-pixel, mixture-of-Gaussian background models," in *Proc. Euro. Conf. Comput. Vis.*, May 2002, pp. 37–49.

[15] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp. 564–577, May 2003.

[16] K. Nummiaro, E. Koller-Meier, and L. Van Gool, "An adaptive color-based particle filter," *J. Image Vis. Comput.*, vol. 21, pp. 99–110, Jan. 2003.

[17] Q. Wei, X. Zhang, C. Li, Y. X. Ouyang, and H. Sheng, "A roubust approach for multiple vehicles tracking using layered particle filter," *Int. J. Electron. Commun.*, vol. 65, pp. 609–618, 2011.

[18] *"TB-6S-LX150T-IMG2 Hardware User Manual,"* Yoshioka, Mar. 2011.

[19] F. Dinechin and B. Pasca, "Floating-point exponential functions for DSP-enabled FPGAs," *Field Program. Technol.*, pp. 110–117, Dec. 2010.

[20] A. Hampapur *et al.*, "Smart video surveillance: Exploring the concept of multiscale spatiotemporal tracking," *IEEE Signal Process. Mag.*, vol. 22, no. 2, pp. 38–51, Mar. 2005.

[21] B. Song *et al.*, "Distributed camera networks: Integrated sensing and analysis for wide-area scene understanding," *IEEE Signal Process. Mag.*, vol. 28, no. 3, pp. 20–31, May 2011.

[22] H. Lani, H. Yin, G. Shrestha, and L. J. Zhang, "Design and implementation of a DSP-based embedded intelligent traffic surveillance system," *J. Commun. Comput. Inf. Sci.*, vol. 226, pp. 221–229, 2011.

[23] T. Tao *et al.*, "A fast block matching algorithm for stereo correspondence," in *Proc. IEEE Int. Cyber. Intell. Syst.*, Sep. 2008, pp. 38–41.

[24] L. Z. Zhu, J. N. Hwang, and H. Y. Cheng, "Tracking of multiple objects across multiple cameras with overlapping and non-overlapping views," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2009, pp. 1056–1060.

[25] M. Wójcikowski, R. Żaglewski, and B. Pankiewicz, "FPGA-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network," *J. Signal Process. Syst.*, vol. 68, no. 1, pp. 1–18, 2012.

[26] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proc. Int. Conf. Pattern Recognit.*, Aug. 2004, vol. 2, pp. 28–31.

[27] M. Valera, "Intelligent distributed surveillance systems: Review," *IEE Vis., Image Signal Process.*, vol. 152, no. 2, pp. 192–204, Apr. 2005.

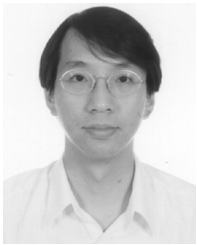
[28] H. Kruegle, *CCTV Surveillance: Analog and Digital Video Practices and Technology*. Burlington, MA: Butterworth-Heinemann, 2006.

[29] W. T. Chen, P. Y. Chen, W. S. Lee, and C. F. Huang, "Design and implementation of a real time video surveillance system with wireless sensor networks," in *Proc. IEEE Veh. Technol. Conf.*, May 2008, pp. 218–222.

[30] J. D. Zhu, L. Yuan, Y. F. Zheng, and R. L. Ewing, "Stereo visual tracking within structured environments for measuring vehicle speed," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 10, pp. 1471–1484, Oct. 2012.

[31] S. C. Chan, H. Y. Shum, and K. T. Ng, "Image-based rendering and synthesis: Technological advances and challenges," *IEEE Signal Process. Mag.*, vol. 24, no. 7, pp. 22–33, Nov. 2007.

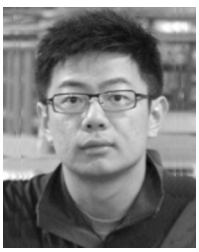
[32] PETS2001 [Online]. Available: ftp://ftp.pets.dg.ac.uk/pub/PETS2001



S. C. Chan (S'87–M'92) received the B.Sc.(Eng.) and Ph.D. degrees from The University of Hong Kong, Pokfulam, Hong Kong, in 1986 and 1992, respectively.

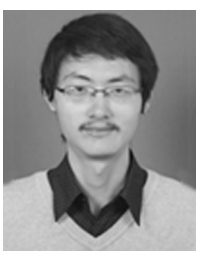
Since 1994, he has been with the Department of Electrical and Electronic Engineering, the University of Hong Kong, where he is currently a Professor. His research interests include fast transform algorithms, filter design and realization, multirate and biomedical signal processing, communications and array signal processing, high-speed A/D converter architecture, bioinformatics, smart grid image-based rendering. He is Associate Editor of the *Journal of Signal Processing Systems* (Springer) and *Digital Signal Processing* (Elsevier).

Dr. Chan is currently a member of the Digital Signal Processing Technical Committee of the IEEE Circuits and Systems Society. He is Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II. He was the Chair of the IEEE Hong Kong Chapter of Signal Processing in 2000–2002, an organizing committee member of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, the 2010 International Conference on Image Processing, and an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I from 2008 to 2009.



Shuai Zhang received the B.Sc.(CE) degree from the Yanshan University, Hebei, China, in 2009, and the M.Sc.(Eng.) degree from The University of Hong Kong in 2010. He is currently working toward the Ph.D. degree in the Department of Electrical and Electronic Engineering, The University of Hong Kong.

From 2010 to 2011, he worked as a Research Assistant with the Department of Electrical and Electronic Engineering, The University of Hong Kong. His research interests focus on multi-modality data fusion, human body tracking, intelligent video surveillance, and statistical video processing.

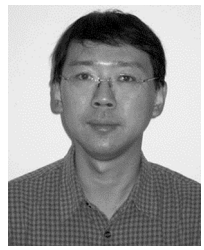


Jia-Fei Wu received the B.S. degree in communications engineering from Jiangxi University of Finance and Economics, Jiangxi, China, in 2010, and the M.S. (Eng.) degree in electrical and electronic engineering, in 2012, from the University of Hong Kong, where he is currently working toward the M.Phil. degree in electrical and electronic engineering.

In 2012, he worked as a Research Assistant with the Department of Electrical and Electronic Engineering, University of Hong Kong. His research interests include image processing and FPGA technology.

Hai-Jun Tan received the B.S. degree in electronic science and technology from South China University of Technology, Guangzhou, China, in 2006. He is currently working toward the M.S. degree at the Department of Communication and Information System, Sun Yat-sen University, Guangzhou, China.

His research interests include hardware implementation of video surveillance and hardware acceleration of 2-D to 3-D conversion algorithms.



J. Q. Ni received the Ph.D. degree in electronic engineering from the University of Hong Kong, in 1998.

He then worked as a postdoc fellow for a joint program between the Sun Yat-Sen University and the Guangdong Institute of Telecommunication Research during 1998 through 2000. Since 2001, he has been with the school of Information Science and Technology, Sun Yat-Sen University, Guangzhou, China, where he is currently a Professor. His research interests include data hiding, digital forensics and image/video processing. He has published more than 50 papers in these areas.

Y. S. Hung received the B.Sc.(Eng.) degree in electrical engineering and the B.Sc. degree in mathematics, both from the University of Hong Kong, and the M.Phil. and Ph.D. degrees from the University of Cambridge, Cambridge, U.K.

He has worked at the University of Cambridge and the University of Surrey before he joined the University of Hong Kong, where he is currently a Professor at the Department of Electrical and Electronic Engineering. He has authored and co-authored over 200 publications in books, journals and conference proceedings. His research interests include computer vision, control systems and biomedical engineering.

Prof. Hung is a Fellow of the Hong Kong Institution of Engineers and the Institution of Engineering and Technology.