Computer Vision and Image Understanding 142 (2016) 65–79

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

Fixed-point Gaussian Mixture Model for analysis-friendly surveillance video coding

Wei Chen[a,b], Yonghong Tian[a,b,c,*], Yaowei Wang[d], Tiejun Huang[a,b,c]

[a] *National Engineering Laboratory for Video Technology, School of EE & CS, Peking University, Beijing, China*
[b] *Cooperative Medianet Innovation Center, China*
[c] *School of Electronic and Computer Engineering, Shenzhen Graduate School at Peking University, Shenzhen, China*
[d] *Department of Electronic Engineering, Beijing Institute of Technology, Beijing, China*

A R T I C L E I N F O

Article history:
Received 4 April 2015
Accepted 20 September 2015
Available online 28 September 2015

Keywords:
Surveillance video
Video coding
Analysis-friendly
Background modeling
Fixed-point Gaussian Mixture Model

A B S T R A C T

With the recent explosion in the use of video surveillance in security, social and industrial applications, it is highly desired to develop "smart" cameras which are capable of not only supporting high-efficiency surveillance video coding but also facilitating some content analysis tasks such as moving object detection. Usually, background modeling is one of fundamental pre-processing steps in many surveillance video coding and analysis tasks. Among various background models, Gaussian Mixture Model (GMM) is considered as one of the best parametric modeling methods for both video coding and analysis tasks. However, a number of floating-point calculations and division operations largely limit its application in the hardware implementation (*e.g.*, FPGA, SOC). To address this problem, this paper proposes a fixed-point Gaussian Mixture Model (fGMM), which can be used in the hardware implementation of the analysis-friendly surveillance video codec in smart cameras. In this paper, we first mathematically derive a fixed-point formulation of GMMs by introducing several integer variables to replace the corresponding float ones in GMM so as to eliminate the floating-point calculations, and then present a division simulation algorithm and an approximate calculation to replace the division operations. Extensive experiments on the PKU-SVD-A dataset show that fGMM can achieve comparable performance with the float GMM on both surveillance video coding and object detection tasks, and outperforms several state-of-the-art methods remarkably. We also implemented fGMM in FPGA. The result shows that the FPGA implementation of our fGMM can process HD videos in real-time, just requiring 140 MHz user logic and 622 MHz DDR3 memory with 64-bit data bus.

1. Introduction

In recent years, surveillance cameras have been widely deployed and video surveillance systems have reached every corner of our modern society. For example, more than 5.9 million surveillance cameras have been deployed in UK; while there are up to more than 30 million surveillance cameras in China. Obviously, with the explosion in the use of video surveillance, it is highly deserved to develop "smart" cameras which, in addition to image capture circuitry, are capable of extracting application-specific information from the captured images [1]. Such smart cameras can be used in a wide range of security, social and industrial applications, including unattended surveillance [1], in-home nanny monitoring [2], road surveillance and traffic control [3], robot guidance and vehicle control [4].

Usually, existing surveillance systems adopt the common video codecs such as H.264/AVC with general settings to compress the captured videos for weeks or months. Some systems even set the compression rate to 300:1 or higher to further reduce network bandwidth and storage capacity. On the other hand, these systems often conduct the video analysis task (*e.g.*, object detection) on the whole frames after decoding the bit-stream from cameras. The better the quality of the frames, the higher the performance of video analysis and search might achieve. Thus a high compression ratio may inevitably influence the accuracy of video analysis. To partially solve this dilemma, as pointed out by [2], the video codec in a "smart" video surveillance system should be analysis-friendly (as Fig. 1 shows), which can not only support high-efficiency surveillance video coding but also facilitates some content analysis tasks such as object detection.

To achieve high-efficiency coding performance, a background-model-based coding method was proposed in [5,6] for surveillance videos, in which the background pictures are modeled from the original input frames so as to provide better reference for encoding the

[*] Corresponding author at: National Engineering Laboratory for Video Technology, School of EE & CS, Peking University, Beijing, China. Fax: +86 10 62751638.
E-mail address: yhtian@pku.edu.cn (Y. Tian).

http://dx.doi.org/10.1016/j.cviu.2015.09.006
1077-3142/© 2015 Elsevier Inc. All rights reserved.

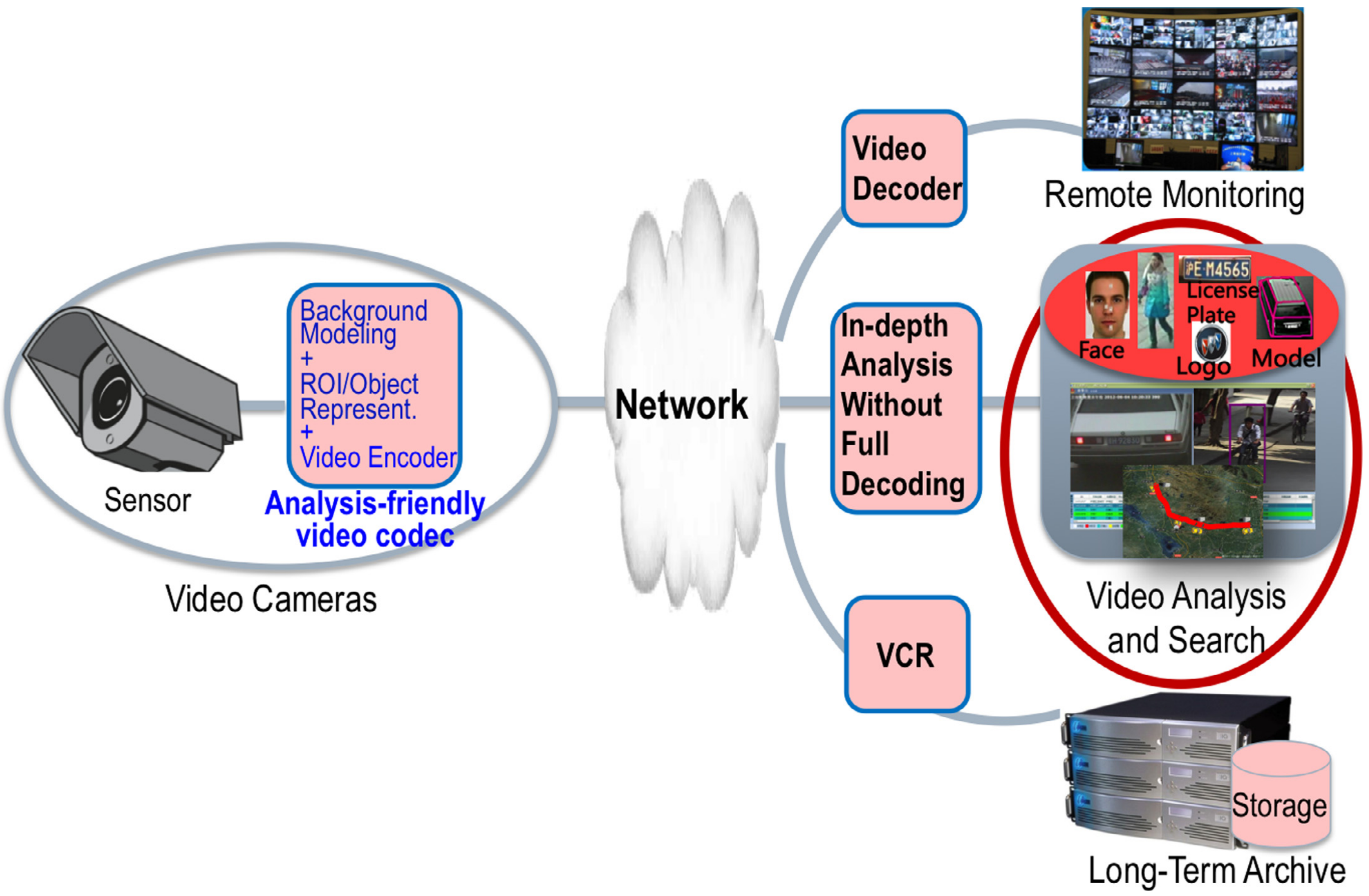

Fig. 1. The conceptual framework of a "smart" video surveillance system with the *analysis-friendly* surveillance video codec in the cameras [2].

following frames, and these background pictures can be encoded into the bitstream as non-display frames to guarantee the decoding match. Experimental results showed that this method could obtain a remarkable compression gain on surveillance videos over H.264/AVC High Profile. It should be noted that this background-model-based coding technology has been adopted in IEEE 1857-S (AVS) and is expected to be implemented in hardware codecs for different video surveillance applications [2]. Meanwhile, background modeling is also one of fundamental pre-processing steps in surveillance video analysis (*e.g.*, [3,7–10]). Many practical evidences have validated that a clean background can be very helpful for analysis tasks such as foreground segmentation and object detection. Therefore, to develop such a smart video surveillance system with the analysis-friendly video codec, it is crucial to design a background modeling method suitable for both video coding and content analysis.

Basically, background modeling in surveillance video coding should be done with low computational complexity and low memory cost. Till to now, two background modeling methods have been proposed in the literature for surveillance video coding. In [11], Zhang et al. proposed a Segment-and-Weight based Running Average (SWRA) method to approximately calculate the background by assigning a larger weight on the frequently-appearing values in the averaging process. As one of low-complexity background modeling methods, it can offer rather good reference for video coding, and more importantly, can be easily implemented in the hardware codec. In [12], Paul et al. proposed to embed the Gaussian Mixture Model (GMM) background generation procedure and the background prediction into H.264/AVC. Compared with SWRA, GMM is computationally complex since it has many floating-point calculations and division operations.

For video analysis, there are many methods in the literature [9], *e.g.*, Frame Differencing, Temporal Median Filtering, Gaussian Mixture Models (GMMs), Bayesian, Kernel Density Estimation (KDE), Codebook methods. Among them, GMM [13–15] is considered as one of the best parametric background modeling methods, due to its good performance and high robustness over different scenes. Recently, many new background modeling algorithms, such as ViBe [16], PBAS [17], SOBS [18], Eigenbackground [19] and Neural-fuzzy model based approach [20], have been proposed. Totally speaking, they show promising performance on several video analysis tasks, and mostly even outperform GMMs. However, ViBe and PBAS are not able to generate backgrounds and thus cannot be embedded into the codec so as to provide prediction reference for video coding, while SOBS and Eigenbackground need massive calculations and thus are difficult to implement in the codecs with low computational complexity. On the other hand, as a variant of the simple running average method, SWRA [11] is not analysis-friendly. It cannot model dynamic background effectively and thus does not satisfy the needs for many video analysis tasks. Thus to the best of our knowledge, GMM should be the best background modeling method for both video coding and video analysis that can be found in the literature till to now.

However, the floating-point calculations and division operations in GMM still present a significant obstacle for its wide application in the hardware implementation of video codecs. For example, if we want to empower the cameras with the analysis-friendly video codec, we need to implement the GMM in FPGA and even SOC. To solve this problem, some recent works (*e.g.*, 21–24) proposed several hardware implementation methods of GMM. Among them, [21] is the newest one and it can process the HD video in real-time. Totally speaking, all these methods adopt the similar strategy by using data conversion to accelerate the processing speed (i.e., using the fixed number of bits to represent the integer and fractional parts). Obviously, this strategy is not an optimization of the GMM model itself. What is more, due to the approximate representation of the fractional parts in the model, the quality of the constructed background will decrease inevitably.

In this paper, we propose a fixed-point Gaussian Mixture Model (fGMM) method for analysis-friendly surveillance video coding. Different from previous works, the proposed fGMM eliminates the floating-point calculations and division operations while being capable of achieving comparable performance. Towards this end, we mathematically derive a fixed-point formulation of GMMs by introducing several integer variables to replace the corresponding float ones in GMM so as to eliminate the float-point calculations. After this, we adopt a division simulation algorithm and an approximate calculation to replace the division operations. Experimental results show that this approximate calculation has little influence on the quality of the modeled background pictures. In addition, our analysis also shows that fGMM saves 46% memory cost compared with its float version.

Extensive experiments were performed on the PKU-SVD-A dataset[1]. This dataset consists of more than 10 videos with different resolutions (ranging from SD, 720p, 1600 × 1200, and 1920 × 1080) and is online publically available. Experimental results show that both in surveillance video coding and object detection tasks, the proposed fGMM can achieve comparable performance with the float GMM, and also outperforms several state-of-the-art methods (*e.g.*, [11,21,24]) remarkably. We also implemented our fGMM in FPGA. The result shows that fGMM has lower hardware requirements (*e.g.*, user logic, memory) to process HD video in real-time, compared with [21].

Our main contributions can be summarized as follows:

(1) A fixed-point formulation of GMMs is mathematically derived by introducing several integer variables to replace the corresponding float ones in GMM so as to eliminate the floating-point calculations. Different from other data conversion methods, the proposed fGMM does not introduce any approximate presentation to convert float into integer.
(2) A division simulation algorithm and an approximate calculation are further proposed for fGMM to replace the division operations. As a result, fGMM can complete all the calculations using integers to achieve comparable performance with the float version of GMM.
(3) Without floating-point calculations and division operations, fGMM can be easily implemented in hardware devices. In our work, fGMM is implemented in FPGA to process HD videos in real-time, just requiring 140 MHz user logic and 622 MHz DDR3 memory with 64-bit data bus.

The remainder of this paper is organized as follows: Section 2 briefly reviews the Gaussian Mixture Model. The proposed fGMM is presented in Section 3. Experimental results are shown in Section 4. Section 5 mainly presents the FPGA implementation of fGMM and Section 6 concludes this paper.

2. Gaussian Mixture Model

Gaussian Mixture Model (GMM), firstly proposed by Stauffer and Grimson [13], is a widely-used parametric background modeling method. Its basic idea is to use several Gaussian distributions to describe a pixel, some of which represent the background in the scene while the others characterize the foreground. To facilitate the online model learning, KaewTraKulPong and Bowden [14] proposed to use Expectation Maximization (EM) to update the parameters. Thus this section briefly reviews the GMM in [14] and then discuss the possible problems when used in hardware implementation.

2.1. GMM

GMM models each pixel by a mixture of K Gaussian distributions. For each Gaussian distribution, there is a weight parameter w representing the time proportion this pixel stays. The greater the value of w is, the longer this pixel stays in this distribution. The Gaussian function is shown as follows:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1}$$

where σ is the standard deviation. A smaller value of σ means a more stable distribution. The background pixel value tends to be the one which stays the longest and keeps static in the video. Therefore, static background tends to form tight clusters in the Gaussian distributions while moving ones form widen clusters. The measurement of this widen or tight cluster is called the fitness value ϕ_k^N. It is calculated by

$$\phi_k^N = \frac{w_k^N}{\sigma_k^N}, \tag{2}$$

where w_k and σ_k are the weight and the standard deviation of the kth Gaussian distribution respectively.

To allow the model to adapt to the changes in the video, an update scheme is applied. In this scheme, every new pixel value is checked through the existing model components in order of fitness, while the first matched Gaussian component will be updated. If it does not find a matched component, a new one will be added with a large standard deviation and a small value of the weighting parameter. The match rule is

$$abs(x - \mu_k) \le Th \times \sigma_k, \tag{3}$$

where $abs(i)$ is the absolute value function, x means the value of the current pixel, μ_k is the mean value of the kth Gaussian distribution, and Th is a threshold value (always setting as 2 or 3). After finding the matched Gaussian distribution, the parameters will be updated using the EM algorithm.

2.2. The EM algorithm

To update the parameters of Gaussian distributions, the EM algorithm [14] begins the estimation of the GMM by using sufficient statistics update equations, then switches to L-recent window version after the first L samples are processed. That is, it uses different updating strategies in different processing stages. When the window size does not reach the threshold L, the EM algorithm can be characterized as follows:

$$w_k^{N+1} = w_k^N + \frac{1}{N+1}\left(p(w_k|x_{N+1}) - w_k^N\right), \tag{4}$$

$$\mu_k^{N+1} = \mu_k^N + \frac{p(w_k|x_{N+1})}{\sum_{i=1}^{N+1} p(w_k|x_i)}\left(x_{N+1} - \mu_k^N\right), \tag{5}$$

$$\Sigma_k^{N+1} = \Sigma_k^N + \frac{p(w_k|x_{N+1})}{\sum_{i=1}^{N+1} p(w_k|x_i)}\left(\left(x_{N+1} - \mu_k^N\right)\left(x_{N+1} - \mu_k^N\right) - \Sigma_k^N\right). \tag{6}$$

Otherwise, the EM algorithm is shown as follows:

$$w_k^{N+1} = w_k^N + \frac{1}{L}\left(p(w_k|x_{N+1}) - w_k^N\right), \tag{7}$$

$$\mu_k^{N+1} = \mu_k^N + \frac{1}{L}\left(\frac{p(w_k|x_{N+1})x_{N+1}}{w_k^{N+1}} - \mu_k^N\right), \tag{8}$$

[1] http://mlg.idm.pku.edu.cn/resources/pku-svd-a.html

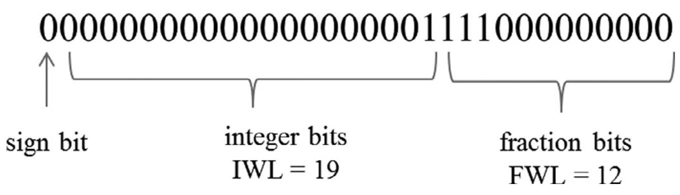

Fig. 2. A sample of floating-point number conversion to a fixed-point number in fpGMM.

$$\Sigma_k^{N+1} = \Sigma_k^N + \frac{1}{L}\left(\frac{p(w_k|x_{N+1})\left(x_{N+1}-\mu_k^N\right)\left(x_{N+1}-\mu_k^N\right)}{w_k^{N+1}} - \Sigma_k^N\right). \tag{9}$$

From (4) to (9), w_k^N and μ_k^N denote the weight and the mean value of the kth Gaussian $\mathcal{M}_k$ respectively when the window size equals to N, Σ_k^N is the square of σ_k^N while σ_k^N means the variance of $\mathcal{M}_k$. x_N is the pixel value of the Nth frame, and $p(w_k|x_{N+1})$ is used to represent whether $\mathcal{M}_k$ is the first match of the current pixel, which can be calculated as follows:

$$p(w_k|x_{N+1}) = \begin{cases} 1, & \mathcal{M}_k \text{ is the first match;} \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

2.3. Discussions

As we can see, the updating methods for variables w, μ and Σ all need the division operations. In the hardware implementation, the division operation is highly time-consuming, much more inefficient than the addition operation. Moreover, in the division operation, all variables must be represented using double or float. This further influences the efficiency of the hardware video codecs. At last, double or float variables need more bytes to store than char or short, so GMM is highly memory-consuming. All these limit its wide application in the hardware implementation of video codecs although it shows promising performance in both video coding and content analysis tasks. To solve these problems, we propose a variant of GMM, called fixed-point GMM (fGMM). It can generate clean background pictures as GMM does. More importantly, it has no division operation or floating-point calculation.

2.4. Data conversions in GMM calculations

To avoid using floating-point calculations in the GMM, different data conversion methods were adopted in previous works. Moon et al. proposed a fixed-point GMM algorithm in [24]. For simplicity, their method is abbreviated as fpGMM in this paper. In fpGMM, each floating-point variable or constant is replaced by a 32-bit integer. It divides a 32-bit integer to three parts: sign, integer and fraction. The numbers of bits assigned to the integer field and the fraction field are called integer word-length (IWL) and fraction word-length (FWL), respectively. For instance, in the case of FWL = 12 and IWL = 19, a floating-point number 1.875 is converted into a fixed-point number 7680 in decimal system (as shown in Fig. 2). Similarly, some other hardware implementations of the GMM (*e.g.*, 21–23) use 8 bits to represent the integer part and 3 bits to represent the fractional part of a double value. Take the number 0.5625 in decimal system (or 0.1001 in binary system) as an example. It becomes 0.5 in decimal system (or 0.100 in binary system) after such a conversion. Obviously, both data conversion methods will bring in heavy approximate error, which further influences the quality of the background frame greatly. In the experiments, we implement the data conversion method in fpGMM and compare the surveillance video coding and analysis results of fpGMM and fGMM. Moreover, we will also discuss the hardware data conversion method in [21] in details and compare the FPGA implementations of [21] and our fGMM.

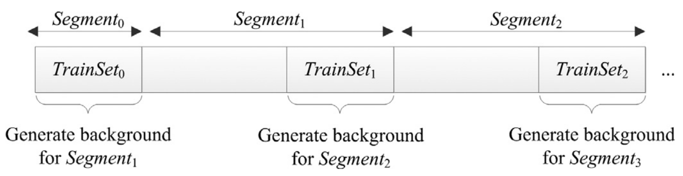
Segment₀
Segment₁
Segment₂
TrainSet₀
TrainSet₁
TrainSet₂
...
Generate background for Segment₁
Generate background for Segment₂
Generate background for Segment₃

Fig. 3. The sequence structure for video coding and object detection.

3. Fixed-point Gaussian Mixture Model

In this section, we first derive some mathematical formulations of fGMM by introducing several integer variables to replace the corresponding float ones in GMM, and then present a division simulation algorithm to replace the division operations. In Section 3.2, we further describe how to conduct the calculations in fGMM using these integer variables, including the matching rule and fitness sorting. Finally, a memory cost analysis is presented in Section 3.3.

3.1. Mathematical derivations

Fig. 3 describes the sequence structure for video coding and object detection [6]. In this structure, several frames labelled as *TrainSet* are used to train the background pictures. That is, an initial group of frames are utilized as $TrainSet_0$ to generate the background picture for $Segment_1$, whereas the last group of frames in $Segment_i$ are utilized as $TrainSet_i$ to generate the background picture for $Segment_{i+1}$ where $i \geq 1$. Then the generated background picture can be used as the reference in video coding or for extracting the foreground mask in object detection on the next segment. In this sequence structure, (4)–(6) are more suitable for GMM parameter updating. Thus, we will conduct some mathematical derivations to them so as to eliminate the divisions and floating-point calculations.

3.1.1. The updating of weight parameter w

The GMM updating method for w is shown in (4). To remove the division by $N+1$ in the right term of (4), we rewrite it by multiplying $N+1$ on both sides, as follows:

$$w_k^{N+1}(N+1) = w_k^N N + w_k^N + p(w_k|x_{N+1}) - w_k^N. \tag{11}$$

Here the term w_k^N can be offset. After this, we introduce a new variable f_k^N to represent the term $w_k^N N$. It means the product of the window size N and the weight parameter of $\mathcal{M}_k$. Then (11) can be simplified as

$$f_k^{N+1} = f_k^N + p(w_k|x_{N+1}). \tag{12}$$

In this way, the updating of w becomes an iterative calculation of the variable f_k^N. As $p(w_k|x_{N+1})$ equals to either 0 or 1, the variable f_k^N in (12) can be represented as an integer.

3.1.2. The updating of mean parameter μ

The GMM updating method for μ is shown in (5). To remove the division by $\sum_{i=1}^{N+1} p(w_k|x_i)$ in the right term of (5), we rewrite it by multiplying $\sum_{i=1}^{N+1} p(w_k|x_i)$ on both sides, as follows:

$$\mu_k^{N+1}\sum_{i=1}^{N+1} p(w_k|x_i) = \mu_k^N \sum_{i=1}^{N+1} p(w_k|x_i) + p(w_k|x_{N+1})\left(x_{N+1}-\mu_k^N\right). \tag{13}$$

As $\sum_{i=1}^{N+1} p(w_k|x_i) = \sum_{i=1}^{N} p(w_k|x_i) + p(w_k|x_{N+1})$, this equation can be transformed as follows:

$$\mu_k^{N+1}\sum_{i=1}^{N+1} p(w_k|x_i) = \mu_k^N \sum_{i=1}^{N} p(w_k|x_i) + \mu_k^N p(w_k|x_{N+1}) + p(w_k|x_{N+1})\left(x_{N+1}-\mu_k^N\right). \tag{14}$$

After offsetting the same term $\mu_k^N p(w_k|x_{N+1})$ in the right side of (14), it can be simplified as follows:

$$\mu_k^{N+1}\sum_{i=1}^{N+1} p(w_k|x_i) = \mu_k^N \sum_{i=1}^{N} p(w_k|x_i) + p(w_k|x_{N+1})x_{N+1}. \quad (15)$$

Then a new variable ν_k^N is introduced to represent the term $\mu_k^N \sum_{i=1}^N p(w_k|x_i)$. It means the product of $\mathcal{M}_k$'s mean value and the matching sum of $\mathcal{M}_k$ in first N frames. Then (15) can be finally simplified as

$$\nu_k^{N+1} = \nu_k^N + p(w_k|x_{N+1})x_{N+1}. \quad (16)$$

3.1.3. The updating of variance parameter Σ

The GMM updating method for Σ is shown in (6). To remove the division by $\sum_{i=1}^{N+1} p(w_k|x_i)$ in the right term of (6), we rewrite it by multiplying $\sum_{i=1}^{N+1} p(w_k|x_i)$ on both sides, as follows:

$$\Sigma_k^{N+1}\sum_{i=1}^{N+1} p(w_k|x_i) = \Sigma_k^N \sum_{i=1}^{N+1} p(w_k|x_i) + p(w_k|x_{N+1})\Big(\big(x_{N+1}-\mu_k^N\big)\big(x_{N+1}-\mu_k^N\big)-\Sigma_k^N\Big). \quad (17)$$

As $\sum_{i=1}^{N+1} p(w_k|x_i) = \sum_{i=1}^{N} p(w_k|x_i) + p(w_k|x_{N+1})$, this equation can be transformed as follows:

$$\begin{aligned}\Sigma_k^{N+1}\sum_{i=1}^{N+1} p(w_k|x_i) = {} & \Sigma_k^N \sum_{i=1}^{N} p(w_k|x_i) \\ & + \Sigma_k^N p(w_k|x_{N+1}) + p(w_k|x_{N+1}) \\ & \times \Big(\big(x_{N+1}-\mu_k^N\big)\big(x_{N+1}-\mu_k^N\big)-\Sigma_k^N\Big). \end{aligned} \quad (18)$$

After offsetting the same term $\Sigma_k^N p(w_k|x_{N+1})$ in the right side of (18), it can be simplified as follows:

$$\begin{aligned}\Sigma_k^{N+1}\sum_{i=1}^{N+1} p(w_k|x_i) = {} & \Sigma_k^N \sum_{i=1}^{N} p(w_k|x_i) \\ & + p(w_k|x_{N+1})(x_{N+1}-\mu_k^N)\big(x_{N+1}-\mu_k^N\big). \end{aligned} \quad (19)$$

Then we introduce a new variable Ψ_k^N to represent the term $\Sigma_k^N \sum_{i=1}^N p(w_k|x_i)$. It means the product of $\mathcal{M}_k$'s variance and the matching sum of $\mathcal{M}_k$ in first N frames. Then (19) can be finally simplified as:

$$\Psi_k^{N+1} = \Psi_k^N + p(w_k|x_{N+1})\big(x_{N+1}-\mu_k^N\big)\big(x_{N+1}-\mu_k^N\big). \quad (20)$$

In this equation, μ_k^N is needed to update Ψ_k^N. As ν_k^N equals to $\mu_k^N \sum_{i=1}^N p(w_k|x_i)$, in order to get the value of μ_k^N, the division operation is unavoidable. It should be noted that the dividend ν_k^N and the divisor $\sum_{i=1}^N p(w_k|x_i)$ are both integers.

To solve the division problem of two integers, [25] utilizes the look-up table and approximate calculations to simulate the division operations. In fact, this is the most efficient method to implement the division operations when the ranges of the dividend and the divisor are limited. However, from the perspective of saving memory, we propose an alternative division simulation algorithm using shift operations. As shown in Algorithm 1, all the variables are integers and the approximate calculation is used. Note that the left and right shift operations in Algorithm 1 are based on powers of 2. Our experiments shown in Section 4 will prove that the approximate calculation has little influence on the constructed background pictures.

Table 1 shows the correspondence of the variables in fGMM and GMM. Note that we do not introduce any extra variables, just using the integer variables to replace the float ones.

Algorithm 1 division using shift operations.

```
1:  procedure DIVISION($integer\ a, integer\ b$)
2:      $integer\ result \leftarrow 0$
3:      while $a >= b$ do
4:          $integer\ tmp \leftarrow 0$
5:          $integer\ i \leftarrow 0$
6:          while $a >= tmp$ do
7:              $a \leftarrow a - tmp$
8:              $result \leftarrow result + (1 << i)$
9:              $tmp \leftarrow tmp >> i$
10:             $i{+}{+}$
11:         end while
12:     end while
13:     return $result$
14: end procedure
```

Table 1
The correspondence of the variables in GMM and fGMM.

Variable in GMM	Variable in fGMM	Equivalence
w_k^N	f_k^N	$f_k^N = w_k^N N$
μ_k^N	ν_k^N	$\nu_k^N = \mu_k^N \sum_{i=1}^N p(w_k\|x_i)$
Σ_k^N	Ψ_k^N	$\Psi_k^N = \Sigma_k^N \sum_{i=1}^N p(w_k\|x_i)$

3.2. Calculations with new variables

In fGMM, the float variables w, μ and Σ are replaced by the corresponding integer variables f, ν and Ψ. However, without the variables w, μ and Σ, some calculations cannot be completed in fGMM, including the Gaussian distribution matching rule shown in (3) and the fitness sorting shown in (2). So this subsection will describe how to accomplish these calculations using the variables f, ν and Ψ.

3.2.1. The matching rule

To determine which Gaussian distribution a new pixel matches, fGMM must calculate the distance between the current pixel value and the mean value of the Gaussian distribution. As shown in (3), if the distance is smaller than the product of the stand deviation and a constant, the pixel is considered to match this Gaussian distribution. Since there are float variables μ_k and σ_k in (3), we conduct some transformations to (3) so that the calculation can be accomplished using variables ν_k and Φ_k^N. As $\Sigma_k^N = (\sigma_k^N)^2$, by squaring both sides, (3) can then be transformed as follows:

$$(x_N - \mu_k^N)^2 \le Th^2 \times \Sigma_k^N. \quad (21)$$

Then multiply $(\sum_{i=1}^N p(w_k|x_i))^2$ at both ends, (21) is turned to the following expression:

$$\left(x_N \sum_{i=1}^{N} p(w_k|x_i) - \mu_k^N \sum_{i=1}^{N} p(w_k|x_i)\right)^2 \le Th^2 \times \Sigma_k^N \sum_{i=1}^{N} p(w_k|x_i) \sum_{i=1}^{N} p(w_k|x_i). \quad (22)$$

As $\mu_k^N \sum_{i=1}^N p(w_k|x_i)$ equals to ν_k^N and $\Sigma_k^N \sum_{i=1}^N p(w_k|x_i)$ equals to Φ_k^N, (22) can be simplified as

$$\left(x_N \sum_{i=1}^{N} p(w_k|x_i) - \nu_k^N\right)^2 \le Th^2 \times \Psi_k^N \sum_{i=1}^{N} p(w_k|x_i). \quad (23)$$

Thus the new matching rule does not need any float variables. All the calculations can be accomplished using integers.

3.2.2. Fitness sorting

In order to find the first matching Gaussian distribution, GMM needs to sort the Gaussian distributions according to the fitness value ϕ_k^N after updating the parameters. This step is also prerequisite of fGMM. As shown in (2), the fitness value ϕ_k^N is represented by float or double because of the division operation in GMM. To avoid the division operations and floating-point calculations, fGMM does not calculate the fitness value ϕ_k^N directly. Instead, it adopts an indirect comparing method using diagonal multiplication to compare two fractions. For instance, if fGMM needs to compare ϕ_k^N of $\mathcal{M}_k$ and ϕ_j^N of $\mathcal{M}_j$, firstly square ϕ_k^N and ϕ_j^N,

$$\left(\phi_k^N\right)^2 = \left(\frac{w_k^N}{\sigma_k^N}\right)^2 = \frac{\left(w_k^N\right)^2}{\Sigma_k^N}, \tag{24}$$

$$\left(\phi_j^N\right)^2 = \left(\frac{w_j^N}{\sigma_j^N}\right)^2 = \frac{\left(w_j^N\right)^2}{\Sigma_j^N}. \tag{25}$$

By multiplying N^2 at both ends of (24) and (25), we can get the following equations:

$$\left(\phi_k^N\right)^2 N^2 = \frac{\left(w_k^N N\right)^2}{\Sigma_k^N} = \frac{\left(f_k^N\right)^2}{\Sigma_k^N}, \tag{26}$$

$$\left(\phi_j^N\right)^2 N^2 = \frac{\left(w_j^N N\right)^2}{\Sigma_j^N} = \frac{\left(f_j^N\right)^2}{\Sigma_j^N}. \tag{27}$$

Note that since N^2 is greater than 0, this multiplication would not change the inequality relationship of ϕ_k^N and ϕ_j^N.

Until now, the inequality relationship of ϕ_k^N and ϕ_j^N still cannot be determined, because the value of Σ_k^N and Σ_j^N cannot be obtained. To transform them to Ψ_k^N and Ψ_j^N, we further multiply $\sum_{i=1}^N p(w_k|x_i)$ in the numerator and denominator of $\frac{(f_k^N)^2}{\Sigma_k^N}$ in (26), and similarly multiply $\sum_{i=1}^N p(w_j|x_i)$ in the numerator and denominator of $\frac{(f_j^N)^2}{\Sigma_j^N}$ in (27). Thus, (26) and (27) can be transformed as follows:

$$\left(\phi_k^N\right)^2 N^2 = \frac{\left(f_k^N\right)^2}{\Sigma_k^N} = \frac{\left(f_k^N\right)^2}{\Sigma_k^N} \times \frac{\sum_{i=1}^N p(w_k|x_i)}{\sum_{i=1}^N p(w_k|x_i)}, \tag{28}$$

$$\left(\phi_j^N\right)^2 N^2 = \frac{\left(f_j^N\right)^2}{\Sigma_j^N} = \frac{\left(f_j^N\right)^2}{\Sigma_j^N} \times \frac{\sum_{i=1}^N p(w_j|x_i)}{\sum_{i=1}^N p(w_j|x_i)}. \tag{29}$$

As $\Psi_k^N = \Sigma_k^N \sum_{i=1}^N p(w_k|x_i)$ and $\Psi_j^N = \Sigma_j^N \sum_{i=1}^N p(w_j|x_i)$, (28) and (29) can be further simplified as follows:

$$\left(\phi_k^N\right)^2 N^2 = \frac{\left(f_k^N\right)^2 \sum_{i=1}^N p(w_k|x_i)}{\Psi_k^N}, \tag{30}$$

$$\left(\phi_j^N\right)^2 N^2 = \frac{\left(f_j^N\right)^2 \sum_{i=1}^N p(w_j|x_i)}{\Psi_j^N}. \tag{31}$$

As such, the inequality $\phi_k^N < \phi_j^N$ equals to

$$\frac{\left(f_k^N\right)^2 \sum_{i=1}^N p(w_k|x_i)}{\Psi_k^N} < \frac{\left(f_j^N\right)^2 \sum_{i=1}^N p(w_j|x_i)}{\Psi_j^N}. \tag{32}$$

To avoid division operations, we can multiply $\Psi_k^N \Psi_j^N$ in both sides of (32). Then the inequality $\phi_k^N < \phi_j^N$ finally equals to

$$\left(f_k^N\right)^2 \sum_{i=1}^N p(w_k|x_i) \Psi_j^N < \left(f_j^N\right)^2 \sum_{i=1}^N p(w_j|x_i) \Psi_k^N. \tag{33}$$

After the derivations, all the variables are integers now and there is no division operation needed in (33).

Table 2
The memory cost of different versions of GMM.

Item	GMM	fGMM	fpGMM
Gaussian number	K	K	K
Pixel number	M	M	M
Memory cost of one Gaussian	26	14	14
Total memory cost	26KM	14KM	14KM

3.3. Analysis of memory cost

In general, compared with other background modeling methods such as SWRA, GMM is much more memory-consuming. The reasons are two-folds. First, GMM needs to buffer one frame in memory and one pixel needs K Gaussian distributions (often setting to 5) to describe it. Supposing there are M pixels in one frame and the memory cost of one Gaussian distribution is S bytes, the total memory cost T is $K \times M \times S$ bytes. Second, one Gaussian distribution needs to store 4 parameters: the weight parameter w, the mean parameter μ, the variance parameter Σ and the matching point counter. In these 4 parameters, the first three parameters are double and the last one is short, so the total memory cost T for GMM is $(8 \times 3 + 2)KM$ bytes, namely 26KM bytes.

In fGMM, we introduce some integer variables to replace the corresponding float ones. This will not lead to the increase of the memory complexity. The reasons are as follows: In fGMM, the pixel number M of one frame and the Gaussian distribution number K are the same with GMM. However, the memory cost of one Gaussian distribution S can be greatly reduced. As the three double parameters, namely weight, mean and variance, can be replaced by integer parameters, the total memory cost T for fGMM can decrease to $(4 \times 3 + 2)KM$ bytes, namely 14KM bytes. That is, compared with GMM, fGMM can save 46% memory cost. For fpGMM, as it just uses 4 bits to represent a double value, the total memory cost T also decreases to $(4 \times 3 + 2)KM$ bytes, namely 14KM bytes. Although achieving the goal of saving memory, the approximate fractional representations sacrifice the quality of the reconstructed background, which is demonstrated in Section 4 in details. The memory costs of different versions of GMM are shown in Table 2.

In practice, we can further reduce the memory cost of fGMM by using less number of Gaussian distributions. Our experimental results in Section 4.3 show that by only using two Gaussian distributions for each pixel, fGMM-2 can also obtain comparable compression efficiency (with only increase of 0.3∼0.4% bit-rate) with GMM-5 (i.e., the GMM with five Gaussian distributions for each pixel), but can save 78.5% memory cost.

4. Experiments

In order to verify the effectiveness of the proposed fGMM, three sets of experiments were conducted: the comparison of different versions of GMM in background qualify, the surveillance video coding experiments on the IEEE 1857-S platform (AVS) [2] when using different background modeling methods, and the object detection experiments to validate whether fGMM can be used to improve the performance of video analysis tasks. Note that IEEE 1857-S is the first video coding standard with the profile for surveillance videos (the jiankong profile in AVS) [2]. To make the experimental results applicable to different video codecs, experiments on the HEVC/H.265 were also conducted.

4.1. Dataset

The experimental sequences are all from the PKU-SVD-A dataset. This dataset consists of more than 10 uncompressed videos with different resolutions (ranging from SD, 720p, 1600 × 1200, and

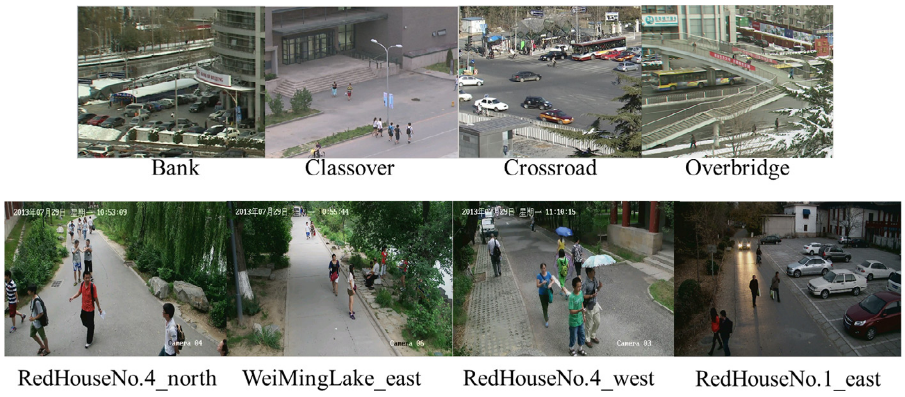

Fig. 4. Sequences used in our experiments. Bank, Classover, Crossroad and Overbridge are SD sequences whose resolution is 720 × 576 while RedHouseNo.4_north, WeiMingLake_east, RedHouseNo.4_west and RedHouseNo.1_east are HD sequences whose resolution is 1920 × 1080.

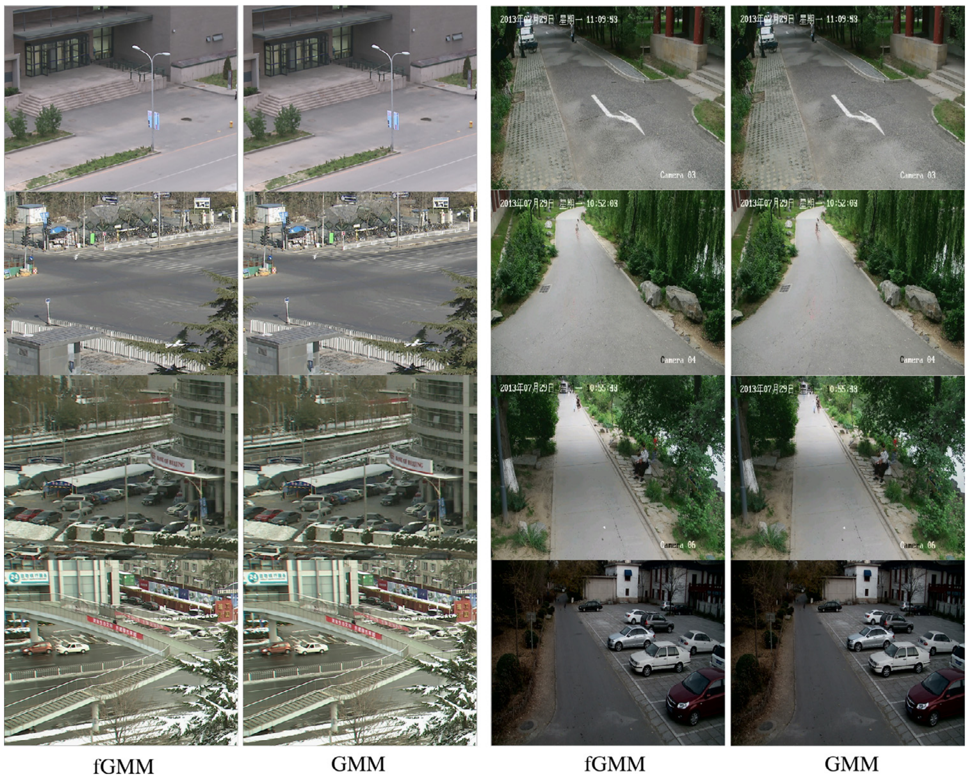

Fig. 5. Some examples of background pictures generated by fGMM and GMM.

1920 × 1080). We adopt this dataset because it is the only public dataset for surveillance video coding till to now, and some sequences from this dataset are also used as IEEE 1857-S surveillance test sequences.

Eight sequences from the PKU-SVD-A dataset were used in our experiments (shown in Fig. 4). In these sequences, four of them are 720 × 576 (SD) sequences and the other four are 1920 × 1080 (HD) sequences. In the SD sequences, the sequences Bank, Crossroad and Overbridge were captured from a very busy crossroad, and contain different kinds of objects such as cars, buses, buildings, pedestrians and so on. The other SD sequence Classover was captured from a campus and can be used to evaluate the methods in the uncomplicated scene. The four HD sequences RedHouseNo.4_north, WeiMingLake_east, RedHouseNo.4_west and RedHouseNo.1_east (abbreviated as No.4_north, Lake_east, No.4_west and No.1_east in this paper) can be used to evaluate the performance of the methods on high resolution videos. Especially, the sequence No.1_east captured in dusk was adopted to evaluate the background modeling methods in extreme illumination conditions. It should be noted that all the eight sequences were used in the video coding experiments, while the three HD sequences No.4_north, Lake_east and No.4_west were used in the object detection experiments, since they are more viable to label the ground-truths (i.e., the rectangles containing objects).

4.2. The background samples of different GMMs

To compare the samples of different GMMs subjectively, we used three methods, namely GMM, fGMM and fpGMM [24], to build the background pictures through training on several frames. The experimental settings are the same as above. Note that for all GMM-related methods in the experiments, we used the same settings of these parameters.

Some examples of the background pictures generated by GMM and fGMM are shown in Fig. 5. We can see that, the background pictures generated by fGMM and GMM are visually almost the same,

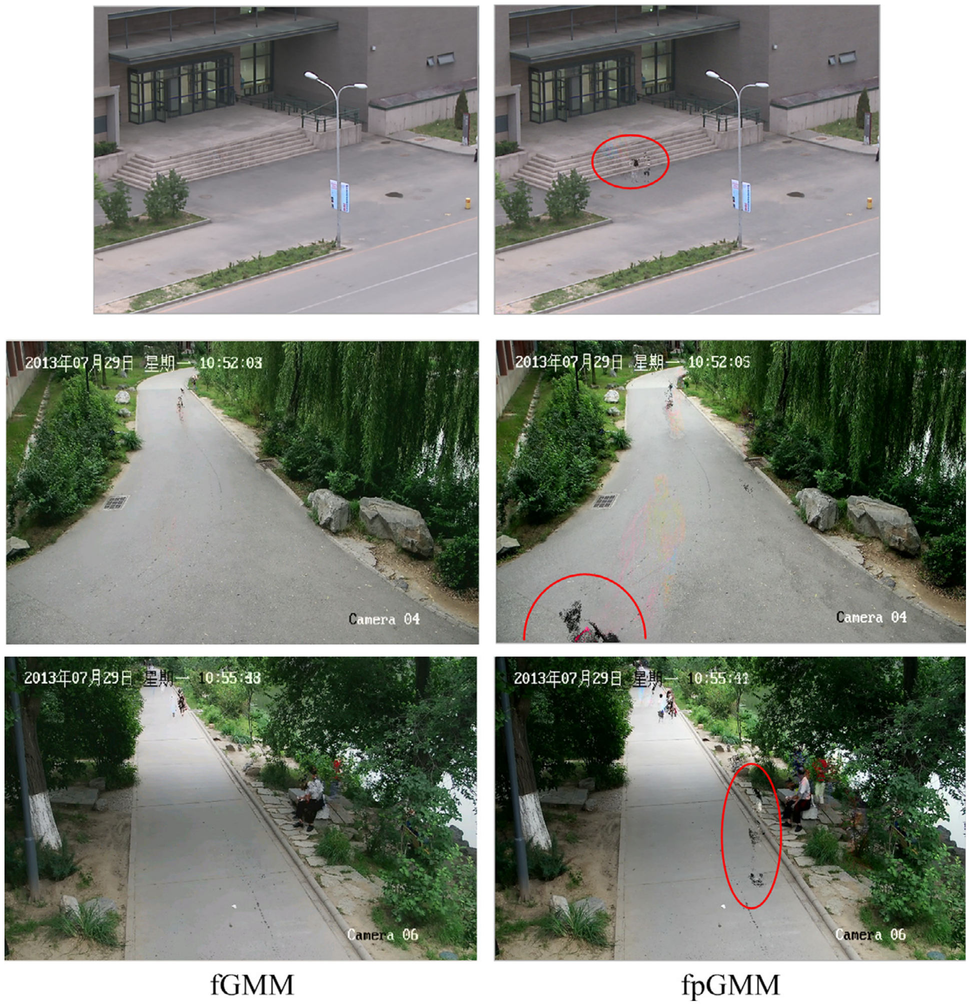
2013年07月29日 星期一 10:52:03
Camera 04
2013年07月29日 星期一 10:52:05
Camera 04
2013年07月29日 星期一 10:55:38
Camera 06
2013年07月29日 星期一 10:55:41
Camera 06
fGMM
fpGMM

Fig. 6. Some examples of background pictures generated by fGMM and fpGMM.

very hard to be distinguished by human eyes. This experimental result confirms that the approximate calculation in fGMM has little influence on the generated background pictures.

In addition, Fig. 6 visualizes some examples of background pictures generated by fGMM and fpGMM. In fpGMM's results, there are several foreground pollution cases in the background pictures (marked by circles) caused by approximate calculations. Since there are many approximate calculations in fpGMM, the overall precision is rather low, exerting negative influence on the quality of background pictures.

4.3. The performance in video coding

Recently, some works [5,6,12,26] have showed that background modeling can significantly improve the performance of surveillance video coding, because the background picture can provide better reference for the following encoding frames. So the best way to assess whether a background modeling method is suitable for surveillance video coding is to evaluate the compressing ratio. Thus this set of experiments were to evaluate the performance of different background modeling methods [27,28] when used in video coding. Five background modeling methods and AVS baseline profile encoder were included in our experiment:

(1) Running Average (RA): a simple pixel-based background modeling method. The background update is as follows:

$$B_{N+1} = \alpha x_N + (1-\alpha) B_N, \qquad (34)$$

where B_N is the background value when the window size equals to N, x_N is the pixel value and α is the updating ratio.

(2) Segment-and-Weight based Running Average (SWRA): SWRA firstly divides pixels at each position in the training frames into several temporal segments, and then calculates their corresponding means and weights. After that, a running and weighted average procedure is used to reduce the influence of foreground pixels. This method is one of the recommended background modeling methods in the AVS reference software.

(3) GMM: The float version of GMM in [14] is used to generate the background pic tures.

(4) Motion Vector Average (MVA): In [29], Vacavant et al. proposed a block-size based background modeling method. Similar to [29], the testing method in this experiment uses the motion vector (MV) in video coding to build the background pictures. MV denotes the direction and distance of the movement of one block. If the size of MV is greater than four, this block is considered to be a foreground block and its pixel value cannot be used to update the background. Otherwise, it uses the current pixel value to update the background.

(5) Fixed-point GMM in [24] (fpGMM): fpGMM uses the float-point conversion to complete the calculations in GMM.

(6) AVS baseline profile encoder (BP): The standard AVS encoder without background modeling.

In our experiment, except the AVS baseline profile encoder, the number of training frames of all the other encoders is 200 and the

Table 3
Configurations of the used SM2_1.6 jiankong profile.

Parameter	Value	Parameter	Value
EntropyCoding	UVLC	FrameRate	25
SearchRange	32	RDO	Used
RateControl	Disable	FME	UMH
RefNumber	32	IntraPeriod	0

background update cycle is 600. The number of all the encoded frames is 1500. For fair comparison, all the anchors and our algorithm were implemented on the SM2_1.6 (very believable AVS reference software) jiankong profile with the common test conditions [2]. The configurations in our experiment are shown in Table 3 and the eight video sequences are encoded with QP = 27, 32, 38 and 45.

The compressing ratios of different methods are measured by bd-rate [30]. The bd-rate results for the eight sequences are shown in Table 4, while the rate-distortion curves are shown in Fig. 7. Compared with the AVS baseline profile encoder (BP), due to the positive influences of the background references, fGMM saves 59.5% bit-rate for the SD sequences and 47.3% bit-rate for the HD sequences on average. These results again validate that background modeling is considerably helpful in surveillance video coding. For the SD sequences, fGMM saves 9.1, 5.8 and 3.7% bit-rate, respectively compared with MVA, RA and SWRA. For the HD sequences, fGMM also saves 9.5, 4.5 and 3.1% bit-rate. These results show that fGMM is more effective than MVA, RA and SWRA when used in video coding. By comparing with fpGMM in [24], fGMM saves 10.0% bit-rate for the SD sequences and 11.7% bit-rate for the HD sequences. At last, compared with GMM, fGMM increases 0.03% bit-rate for the SD sequences and 0.1% bit-rate for the HD sequences. This performance loss is mainly caused by the approximate calculation in fGMM. In practice, it is almost ignorable.

To evaluate the performance of fGMM on different encoders, an additional experiment based on HM12.0 (the HEVC/H.265 reference software) was conducted. In [31], a method which utilizes the background pictures for surveillance video coding and prediction in HEVC was proposed. We implemented this approach in our experiments and used GMM and fGMM to generate the background pictures so as to compare their performance. The configurations for HEVC/H.265 are shown in Table 5 and the eight video sequences are encoded with QP at 22, 27, 32 and 37. The experimental results are shown in Table 6. Since the HEVC encoder supports the variable block structure, the results are a little bit different with those on AVS. But no matter on HEVC or AVS, the performance loss is both considerably small and thus can be acceptable under the conditions of using approximate calculations in fGMM.

In the experiments above, the number of Gaussian distributions equals to 5. We conducted an additional experiment using fGMM-2 to build the background pictures for video coding. In this experiment, all the parameters are the same with above except the number of Gaussian distributions. The results are shown in Table 7. Compared with GMM-5, fGMM-2 increases 0.3% bit-rate on SD and 0.4% bit-rate on HD on average. Taking the memory saving brought by fGMM-2 into account, the bit-rate increase is rather acceptable.

4.4. The performance in object detection

The purpose of this set of experiments was to evaluate the performance of fGMM when used in video analysis tasks (*e.g.*, object detection). In the experiment, the quantitative evaluation was done by comparing the similarity between the subtraction results and the ground-truths. Often, the ground-truths should be manually labelled in the form of accurate contours of all foreground objects. However, labeling such ground-truths on a large data set is very labor-consuming. A simplified method is proposed in [32] and this evaluation method has been used in many papers (*e.g.* [19]), where some random frames are extracted from the test data, and the ground-truths of these frames are generated by labeling the bounding boxes of foreground objects. When at least 40% pixels in an object bounding box (note that 25% was used in [32]) are determined as foregrounds by an algorithm, that object is viewed as a correct detection. The true positive TP is defined as the ratio of the number of correctly detecting objects vs. the number of labelled objects in the ground-truths, and the negative positive FP is measured as the percentage of objects outside the bounding box that are incorrectly classified as foreground [32]. For different background modeling methods, we can plot the ROC curves for each method under different thresholds, and then calculate the Area Under the ROC Curve (AUC). AUC measures the performance from different thresholds and then calculates a single overall score. The geometric meaning of AUC is the area between the ROC curve and the *x*-axis.

To conduct these experiments, we implemented a Smart Camera system proposed in [2]. The framework and backend interface are shown in Figs. 8 and 9, respectively. This Smart Camera is a real-time surveillance video processing system that can implement the background-model-based surveillance video coding and moving foreground detection inside the camera. Its input sequence is captured from a camera and the output contains four windows shown in Fig. 9. The top-left is the decoded frames, while the top-right is the background frames which can be used in video coding and video analysis at the same time. The bottom-left is foreground masks generated by background subtraction and the bottom-right is the object detection results that are obtained just using the foreground detection.

In this experiment, the performance of object detection in both processing speed and accuracy can be improved remarkably by utilizing the background pictures generated in the coding process. To evaluate the object detection performance, the well-known

Table 4
Compressing ratio saving of fGMM compared with different background modeling methods on AVS.

Sequence (SD)	Bank (%)	Cross-road (%)	Class-over (%)	Over-bridge (%)	Average (%)
fGMM vs. SWRA	−4.2	−2.3	−6.5	−1.8	**−3.7**
fGMM vs. GMM	0.1	0.0	0.0	0.0	**0.03**
fGMM vs. RA	−7.8	−5.2	−2.6	−7.4	**−5.8**
fGMM vs. MVA	−11.4	−6.4	−7.8	−10.7	**−9.1**
fGMM vs. BP	−63.4	−34.4	−76.9	−63.2	**−59.5**
fGMM vs. fpGMM	−6.1	−8.7	−11.7	−13.6	**−10.0**
Sequence (HD)	No.4-west (%)	No.4-north (%)	Lake-east (%)	No.1-east (%)	Average (%)
fGMM vs. SWRA	−2.5	−3.1	−2.4	−4.5	**−3.1**
fGMM vs. GMM	0.2	0.1	0.1	0.0	**0.1**
fGMM vs. RA	−6.1	−5.5	−4.9	−1.5	**−4.5**
fGMM vs. MVA	−13.6	−7.3	−3.0	−14.2	**−9.5**
fGMM vs. BP	−51.1	−41.8	−19.8	−76.6	**−47.3**
fGMM vs. fpGMM	−9.6	−13.4	−8.4	−15.5	**−11.7**

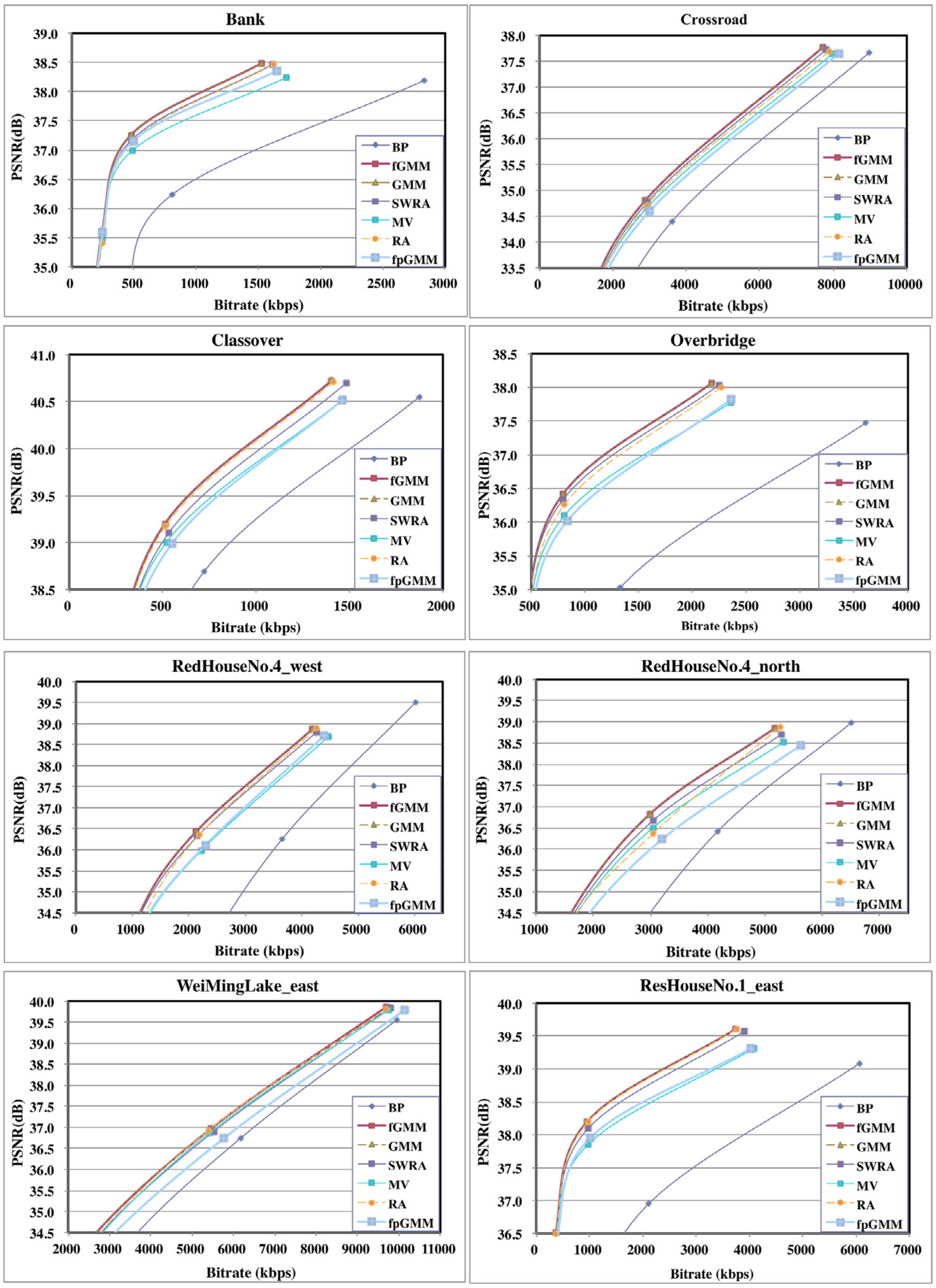

Fig. 7. Rate-distortion curves for the eight sequences on AVS. The top row lists the SD sequences: Bank, Crossroad, Classover and Overbridge. The bottom row lists the HD sequences: RedHouseNo.4_west, RedHouseNo.4_north, WeiMingLake_east and RedHouseNo.1_east.

Table 5
Configurations of the used HM 12.0 main profile.

Parameter	Value	Parameter	Value
EntropyCoding	CABAC	FrameRate	25
SearchRanged	64	RDO	Used
RateControl	Disable	Profile	Main
MaxCUheight	64	MaxCUwidth	64
RefNumber	4	IntraPeriod	−1

discriminatively trained part-based models (DPM) [33] is used in the video analysis module to get the final object detection results. It is based on mixtures of multi-scale deformable part models to represent highly variable object classes, thus achieving better performance than other state-of-the-art methods.

Six methods were used in this experiment to evaluate their performances in terms of the object detection task. The first method does not have the background modeling and the background subtraction module in Fig. 8, but just completes the detection processes on the whole decoded video frames (directly denoted as DPM_{Frame} here). The another three methods are to utilize the DPM with three different background modeling methods: fGMM, GMM and SWRA (denoted as DPM_{fGMM}, DPM_{GMM} and $DPM_{SWRA,}$ respectively). The last two are to utilize the DPM with ViBe (denoted as DPM_{ViBe}) and PBAS (denoted as DPM_{PBAS}). Note that we directly used the segmentation results of ViBe and PBAS as the input of DPM, because it can only obtain the foregrounds. In all GMM-related methods, they exactly follow the framework in Fig. 8: in the encoding side, the input videos are encoded with background modeling, and the background pictures are also encoded

Table 6
Compressing ratio saving of fGMM compared with GMM on HEVC.

Sequence(SD)	Bank (%)	Cross-road (%)	Class-over (%)	Over-bridge (%)	Average (%)
fGMM vs.GMM	0.1	0.0	0.0	0.0	**0.03**
Sequence(HD)	No.4-west (%)	No.4-north (%)	Lake-east (%)	No.1-east (%)	Average (%)
fGMM vs. GMM	0.1	0.2	0.1	0.0	**0.1**

Table 7
Compressing ratio saving of fGMM-2 compared with GMM-5.

Sequence(SD)	Bank (%)	Cross-road (%)	Class-over (%)	Over-bridge (%)	Average (%)
fGMM-2vsGMM-5	0.5	0.1	0.1	0.5	**0.3**
Sequence(HD)	No.4-west (%)	No.4-north (%)	Lake-east (%)	No.1-east (%)	Average (%)
fGMM-2vsGMM-5	0.2	0.4	0.8	0.2	**0.4**

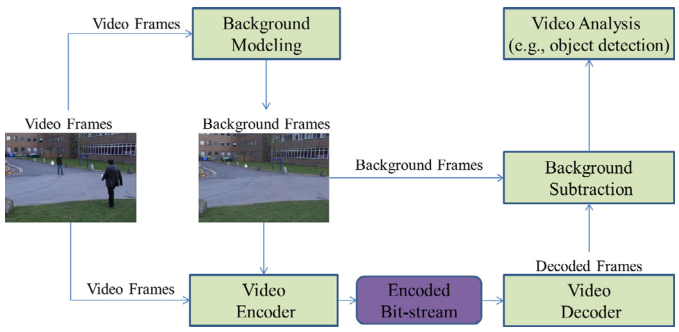

Fig. 8. The framework of object detection by utilizing the background pictures generated in the coding process.

into the bit-stream; while in the decoding side, the bit-streams are decoded to get the video frames and meanwhile the decoded background pictures can be utilized to extract the foreground regions; finally, object detection with DPM is conducted on those foreground regions. Noted that in the whole procedure of the three GMM-related methods, all the steps are identical except that different background modeling methods are used.

The ROC curves of different methods are shown in Fig. 10. We can see that DPM_{fGMM}, DPM_{GMM} and DPM_{SWRA} all outperform the DPM_{Frame}. The reason lies that the foreground regions obtained by background subtraction have excluded some areas where the foreground objects will not appear. For example, in the left frame in Fig. 11, the yellow rectangles are foreground regions generated by fGMM-based background subtraction and the green rectangles are the final detection results; while in the right frame in Fig. 11, the red rectangles are the detection results when DPM_{Frame} is performed directly on the whole decoded frame. There is a false detection rectangle on the upper right corner in the right frame, where the leaves are regarded as a pedestrian. However, in the left side, since the fGMM-based detection is just performed inside the yellow box, this false detection can be avoided. We also notice that DPM_{fGMM} and DPM_{GMM} have nearly the same performance in most cases, both better than DPM_{SWRA}. Basically, the DPM method is very sensitive to foreground regions. That is, one or two pixels deviation of the foreground region may result in a great difference of the final result. This is the reason why the DPM_{fGMM} curve and the DPM_{GMM} curve are separate in

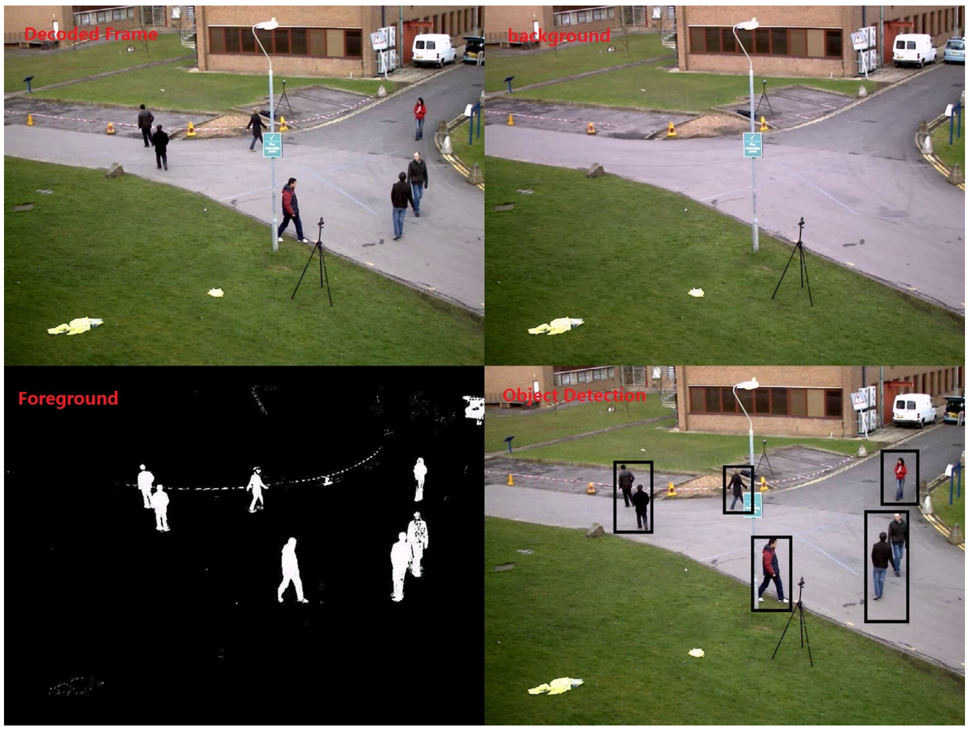

Fig. 9. The backend interface of our Smart Camera system.

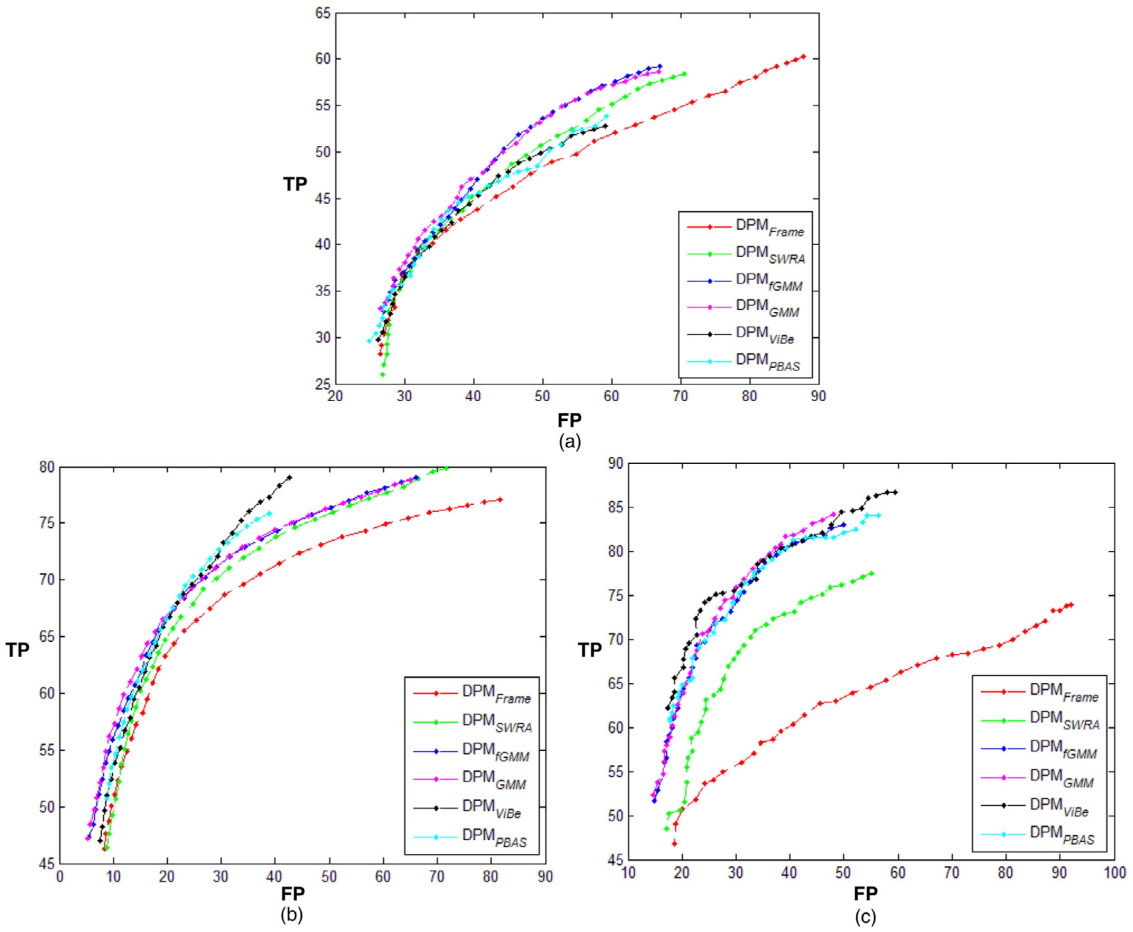

Fig. 10. ROC curves of different methods on PKU-SVD-A dataset. (a) RedHouseNo.4_west, (b) RedHouseNo.4_north, and (c) WeiMingLake_east.

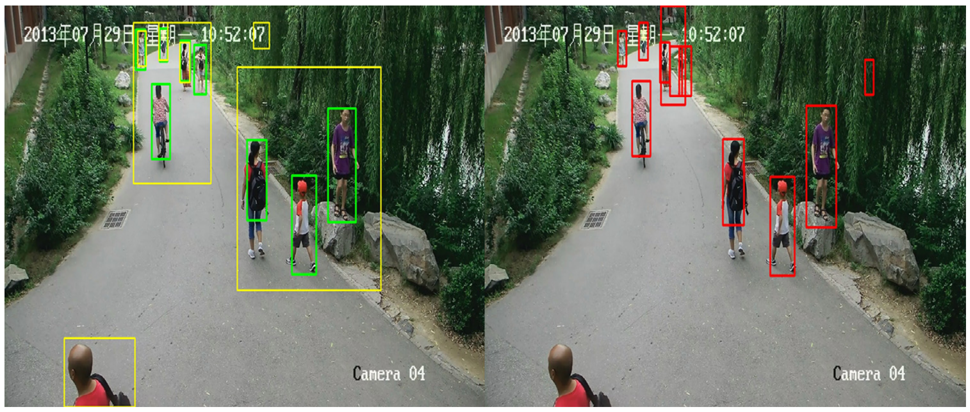
2013年07月29日 星期一 10:52:07
Camera 04
2013年07月29日 星期一 10:52:07
Camera 04

Fig. 11. An example of object detection. The left frame shows the detection results of DPM$_{fGMM}$, where the yellow rectangles denote the foreground region generated by background subtraction and the green rectangles denote the detection results. The right frame depicts the detection results when the DPM$_{Frame}$ is directly performed on the whole decoded frame, where the red rectangles denote the detection results. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

some parts. The AUCs of different background modeling methods are shown in Table 8. They are consistent with the ROC curves.

From Table 8, we can see that ViBe and PBAS are slightly better than the GMM-related methods in most cases. However, they cannot be used in video coding. Therefore, fGMM is still one of the best background modeling methods for the hardware implementation of the analysis-friendly video codec.

Also, the detection time results of the six DPM methods on a 4-core PC with 3.3GHz CPU and 4G RAM are recorded in Table 9. Since the DPM$_{Frame}$ has to scan the whole frame, its speed is rather slow, more than 40 seconds per frame for 1920 × 1080 sequences on average. While the DPMs with background modeling methods (i.e., DPM$_{fGMM}$, DPM$_{GMM}$ and DPM$_{SWRA}$) are much faster. The main reason is that the detection is only performed on foreground regions, thus saving a lot of time. Note that the DPM$_{SWRA}$ is slower than DPM$_{fGMM}$ and DPM$_{GMM}$. This is because the foreground pollution in the background pictures generated by SWRA results in larger foreground regions, thus increasing the detection time of DPM$_{SWRA}$.

Table 8
AUCs of the DPMs with different background modeling methods.

Sequence	No.4_west	No.4_north	Lake_east
DPM$_{fGMM}$	0.503	0.745	0.754
DPM$_{GMM}$	**0.508**	0.746	0.765
DPM$_{SWRA}$	0.482	0.724	0.703
DPM$_{Frame}$	0.439	0.695	0.580
DPM$_{ViBe}$	0.487	**0.761**	**0.772**
DPM$_{PBAS}$	0.479	0.753	0.746

Table 9
Detection time of the DPMs with different background modeling methods.

Sequence	No.4_west	No.4_north	Lake_east
DPM$_{fGMM}$	8.14s	11.87s	9.27s
DPM$_{GMM}$	8.37s	11.93s	9.21s
DPM$_{SWRA}$	17.05s	23.32s	10.91s
DPM$_{Frame}$	44.17s	40.73s	46.77s
DPM$_{ViBe}$	7.82s	10.46s	9.01s
DPM$_{PBAS}$	8.12s	10.92s	11.14s

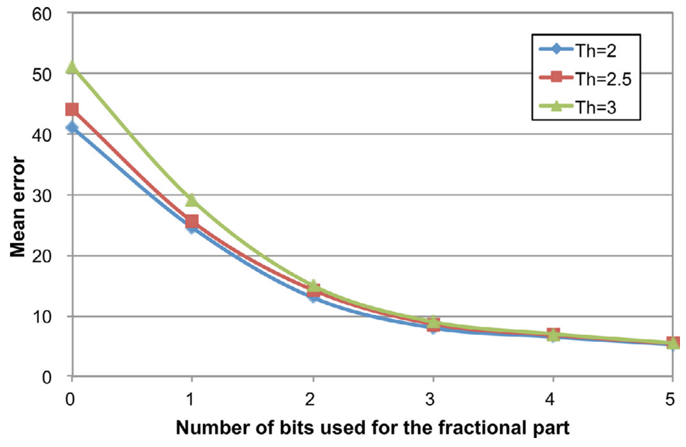

Fig. 12. The mean representation error, depending on the number of bits for the fractional part for different values of the parameter α.

5. The FPGA implementation of fGMM

As discussed in Section 2.4, most existing works (*e.g.*, [21–23]) use data conversions to make the GMM more suitable for the hardware implementation. Different from them, the fGMM needs no data conversions because of the mathematical derivations presented in Section 3. Thus no approximate representation will be adopted in our method.

In [21], a simple experiment was conducted on the running average method to evaluate the influence of different fixed-point precisions on the background model. In this paper, a similar experiment is performed on the GMM. For simplicity, the number of Gaussians is set to 1. The matching rule and update strategy for Gaussian parameters (presented in details in Sections 2.1 and 2.2) are as follows:

$$abs(x - \mu_k) \le Th \times \sigma_k, \tag{35}$$

$$\mu_k^{N+1} = \mu_k^N + \frac{p(w_k|x_{N+1})}{\sum_{i=1}^{N+1} p(w_k|x_i)}(x_{N+1} - \mu_k^N), \tag{36}$$

$$\Sigma_k^{N+1} = \Sigma_k^N + \frac{p(w_k|x_{N+1})}{\sum_{i=1}^{N+1} p(w_k|x_i)}((x_{N+1} - \mu_k^N)(x_{N+1} - \mu_k^N) - \Sigma_k^N). \tag{37}$$

For several pre-selected values of *Th* (2, 2.5, 3) and 0∼5 bits for the fractional part, (36) was calculated for all possible input values. Then the mean error between the examined representation and double floating point precision was calculated. The result is presented in Fig. 12. We can see, the more bits it used for the fractional part, the less the error is. But using more bits means higher memory costs. To balance the memory costs and errors brought by approximate representations, most of existing methods (*e.g.*, [21–23]) use 3 bits to represent the fractional part. This will influence the quality of the background frame greatly. For fGMM, we can directly implement it on hardware devices with few approximate calculations and lower hardware requirements by comparison with [21]. The framework and physical map of the FPGA implementation of our fGMM are shown in Figs. 13 and 14, respectively.

In [21], the one-model size (namely, the number of bits required to model a pixel using one Gaussian) is 47 bits. In the FPGA implementation of fGMM, it is 48 bits (16 bits for the mean value, 8 bits for the weight and 24 bits for the variance). In fact, we can further decrease the one-model size to 40 bits by just using 16 bits for the variance. But this means that some approximate representation should be adopted like [21]. Therefore, to guarantee the quality of the reconstructed background frame, the one-model size is set to 48 bits in our FPGA implementation. In [21], the processing speed can reach 60 fps for HD videos (1920 × 1080) when the hardware configurations are as follows: the user logic works with 200 MHz clock, the 64-bit data bus to DDR3 memory whose data rate is 800 MHz. In the FPGA implementation of our fGMM, we just requires 140 MHz user logic and 622 MHz DDR3 memory. The calculations are as follows: In our design, we combine two pixels as a group. With this strategy, only 9 cycles are required to update the GMM parameters for two pixels, so the required user logic is 1920 × 1080 × 60 × (9/2) = 560 MHz. As we have 4 processing units (PUs) in the background update module, the actual required user logic is 560/4 = 140 MHz. For the DDR3 memory,

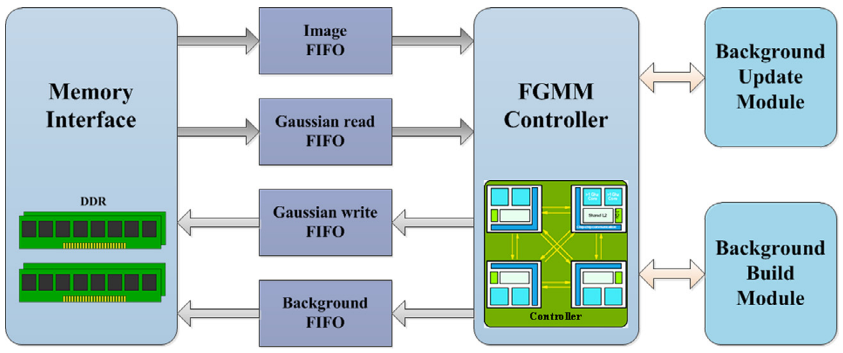

Fig. 13. The framework of the FPGA implementation of our fGMM, where FIFO denotes the first-in-first-out buffer.

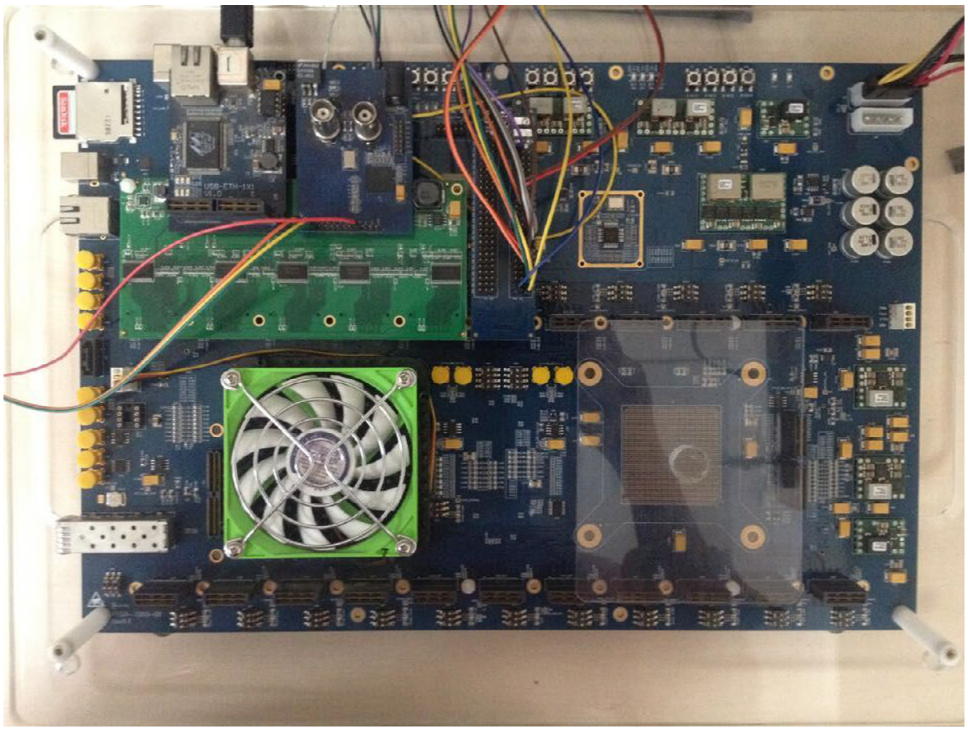

Fig. 14. The physical map of the FPGA implementation of fGMM.

Table 10
Required hardware configurations for real-time processing for fGMM and [21].

Method	One-model size	Data bus	User logic	DDR clock
[21]	47bits	64-bit	200 MHz	800 MHz
fGMM	48bits	64-bit	140 MHz	622 MHz

the total bandwidth is $1920 \times 1080 \times 60 \times (2 \times 2 \times 6) = 2.986$ GBps $= 23.888$ bits, where the first “2” in brackets means accessing the memory two times, for writing and reading memory, the second “2” in brackets represents the number of Gaussian distributions, while the number “6” in brackets means the cycles we need to process a pixel using pipeline. Assuming the bandwidth utilization is 60%, the required frequency for DDR3 memory is $23.888/64/0.6 = 0.622$GHz $= 622$ MHz. Therefore, we can use a 622 MHz DDR3 memory to realize real-time processing. The requirements of hardware devices for fGMM and [21] are shown in Table 10.

More importantly, the background frames generated by [21] are obviously worse than those by fGMM. From Fig. 4 in [21], we can see that its background frame contains some visible foreground pollutions brought by converting float into integers. However, as shown in Fig. 5, few foreground pollutions can be found in the background frames generated by our fGMM.

6. Conclusion

This paper proposes a fixed-point Gaussian Mixture Model (fGMM) which eliminates the floating-point calculations and division operations so that it can be used in the hardware implementation of the analysis-friendly surveillance video codec. Compared with other data conversion methods, fGMM has three remarkable advantages. First, fGMM does not introduce any approximate representation to convert float into integer, and also avoids division operations by a division simulation algorithm and an approximate calculation. Extensive experiments on various video coding and content analysis tasks show that fGMM can complete all the calculations using integers to achieve comparable performance with the float version of GMM. Second, fGMM greatly cuts the memory cost due to the removal of floating-point calculations. Experimental results also show that fGMM can saves 46% memory cost compared with its float version. At last, fGMM can be easily implemented in hardware devices. Compared with the state-of-the-art method [21], fGMM has much lower requirements for real-time HD video processing. In the future, we will implement it in SOC so as to effectively empower the cameras with such a hardware codec.

Acknowledgment

This work is partially supported by a grant from the National Basic Research Program of China under 2015CB351806, and partially by grants from the National Natural Science Foundation of China under contract no. 61390515, no. 61425025, no. 61421062 and no. 61471042, and Y.H. Tian and T.J. Huang are also supported by Shenzhen Peacock Plan.

References

[1] H. Arun, B. Lisa, Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking, IEEE Signal Proc. Mag. 22 (2) (2005) 38–51.
[2] W. Gao, Y. Tian, T. Huang, S. Ma, X. Zhang, IEEE 1857 standard empowering smart video surveillance systems, IEEE Intell. Syst. 29 (5) (2014) 1–10.
[3] T. Celik, H. Kusetogullari, Solar-powered automated road surveillance system for speed violation detection, IEEE Trans. Ind. Electron 57 (9) (2010) 3216–3227.
[4] C. Tsugawa, Vision-based vehicles in japan: machine vision systems and driving control systems, IEEE Trans. Ind. Electron 41 (3) (1994) 398–405.
[5] X. Zhang, L. Liang, Q. Huang, Y. Liu, T. Huang, W. Gao, An efficient coding scheme for surveillance videos captured by stationary cameras, in: Proceedings of International Conference on Visual Communications Image Processing, 2010, p. 77442A.
[6] X. Zhang, T. Huang, Y. Tian, W. Gao, Background-modeling based adaptive prediction for surveillance video coding, IEEE Trans. Image Process. 23 (2) (2014) 769–784.
[7] L. Zhao, X. Zhang, Y. Tian, R. Wang, T. Huang, A background proportion adaptive lagrange multiplier selection method for surveillance video on HEVC, in: Proceedings of International Conference on Multimedia & Expo, 2013, pp. 1–6.
[8] X. Guo, S. Li, X. Cao, Motion matters: a novel framework for compressing surveillance videos, in: Proceedings of the 21st ACM International Conference on Multimedia, 2013, pp. 549–552.

[9] T. Bouwmans, Traditional and recent approaches in background modeling for foreground detection: an overview, Comput. Sci. Rev. 11–12 (2014) 31–66.
[10] R. Canals, A. Roussel, J.-L. Famechon, S. Treuillet, A biprocessor-oriented vision-based target trackingsystem, IEEE Trans. Ind. Electron 49 (2) (2002) 500–506.
[11] X. Zhang, Y. Tian, T. Huang, W. Gao, Low-complexity and high-efficiency background modeling for surveillance video coding, in: Proceedings of International Conference Visual Communications and Image Processing, 2012, pp. 1–6.
[12] M. Paul, W. Lin, C.T. Lau, B.-s. Lee, Video coding using the most common frame in scene, in: Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing, 2010, pp. 734–737.
[13] C. Stauffer, W.E.L. Grimson, Adaptive background mixture models for real-time tracking, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 1999.
[14] P. KaewTraKulPong, R. Bowden, An improved adaptive background mixture model for real-time tracking with shadow detection, in: Proceedings of the 2nd European Workshop on Advanced Video-Based Surveillance Systems, 2002.
[15] E. Hayman, J.-O. Eklundh, Statistical background subtraction for a mobile observer, Nice, France, 2003, pp. 67–74.
[16] O. Barnich, M. Van Droogenbroeck, Vibe: a powerful random technique to estimate the background in video sequences, in: Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing, 2009, pp. 945–948.
[17] M. Hofmann, P. Tiefenbacher, G. Rigoll, Background segmentation with feedback: the pixel-based adaptive segmenter, in: Proceedings of IEEE Conference on Computer Vision Pattern Recognition Workshops, 2012, pp. 38–43.
[18] L. Maddalena, A. Petrosino, The 3dSOBS+ algorithm for moving object detection, Comput. Vis. Image Underst. 122 (2014) 65–73.
[19] Y. Tian, Y. Wang, Z. Hu, T. Huang, Selective eigenbackground for background modeling and subtraction in crowded scenes, IEEE Trans. Circuits Syst. Video Technol. 23 (11) (2013) 1849–1864.
[20] M.I. Chacon-Murguia, S. Gonzalez-Duarte, An adaptive neural-fuzzy approach for object detection in dynamic backgrounds for surveillance systems, IEEE Trans. Ind. Electron 59 (8) (2012) 3286–3298.
[21] T. Kryjak, M. Komorkiewicz, M. Gorgon, Real-time background generation and foreground object segmentation for high-definition colour video stream in FPGA device, J. Real-Time Image Process. 9 (1) (2014) 61–77.
[22] K. Appiah, A. Hunter, A single-chip FPGA implementation of real-time adaptive background model, in: Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology, 2005, 2005, pp. 95–102.
[23] H. Jiang, H. Ardo, V. Owall, Hardware accelerator design for video segmentation with multi-modal background modelling, in: Proceedings of IEEE International Symposium on Circuits and Systems. ISCAS 2005, 2005, pp. 1142–1145.
[24] Y. Moon, C. Leung, K. Pun, Fixed-point GMM-based speaker verification over mobile embedded system, in: Proceedings of ACM SIGMM Workshop on Biometrics Methods and Application, 2003, pp. 53–57.
[25] M. Shi, A. Bermak, An efficient digital vlsi implementation of gaussian mixture models-based classifier, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (2006) 962–974.
[26] X. Zhang, L. Liang, Q. Huang, T. Huang, W. Gao, A background model based method for transcoding surveillance videos captured by stationary camera, in: Proceedings of Picture Coding Symposium, 2010, pp. 78–81.
[27] B. Thierry, F.E. Baf, B. Vachon, Background modeling using mixture of gaussians for foreground detection c a survey, in: Recent Patents on Computer Science, 2008, pp. 219–237.
[28] S. Andrews, V. Antoine, A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos, Comput. Vis. Image Und. 122 (2014) 4–21.
[29] A. Vacavant, L. Robinault, S. Miguet, C. Poppe, R. van de Walle, Adaptive background subtraction in h.264/avc bitstreams based on macroblock sizes, in: Proceedings of International Conference on Computer Vision Theory and Applications, 2011.
[30] G. Bjontegard, Calculation of average PSNR differences between rd-curves, in: ITU-T VCEG-M33, ITU-T, 2001.
[31] X. Zhang, Y. Tian, T. Huang, S. Dong, W. Gao, Optimizing the hierarchical prediction and coding in HEVC for surveillance and conference videos with background modeling, IEEE Trans. Image Process. 23 (10) (2014) 4511–4526.
[32] B. Klare, S. Sarkar, Background subtraction in varying illuminations using an ensemble based on an enlarged feature set, in: Proceedings of IEEE Conference on Computer Vision Pattern and Recognition Workshops, 2009, pp. 66–73.
[33] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, D. Ramanan, Object detection with discriminatively trained part-based models, IEEE Trans. Pattern Anal. Mach. Intell. 32 (9) (2010) 1627–1645.