



# Fast background subtraction with adaptive block learning using expectation value suitable for real-time moving object detection

Vince Jebryl Montero<sup>1</sup> · Woo-Young Jung<sup>1</sup> · Yong-Jin Jeong<sup>1</sup>

Received: 11 April 2019 / Accepted: 2 December 2020 / Published online: 4 January 2021  
© The Author(s), under exclusive licence to Springer-Verlag GmbH, DE part of Springer Nature 2021

## Abstract

This paper presents a method of moving object detection through a fast background subtraction technique suitable for real-time performance in wide range of platforms. An intermittent background update using adaptive blocks individually calculates the learning rate through expected difference values. Then, coupled with a fast background subtraction process, the design achieves fast throughput with well-rounded performance. To compensate for the lagging effects of intermittent background update, an adaptation bias is devised to improve precision and recall metrics. Experiments show a versatile performance in varying scenes with overall results better than conventional techniques. The proposed method achieved a fast execution speed of up to 56 fps in PC using Full HD video. It also achieved 655 fps and 83 fps in PC and ARM core-embedded platform, respectively, using the minimum input resolution of  $320 \times 240$ . Overall, it is suitable for real-time performance applications.

**Keywords** Background subtraction · Moving object detection · Background modeling

## 1 Introduction

In modern surveillance systems, cameras are often equipped with capabilities to gather visual cues and useful information from their surroundings. Such capabilities are used for a wide range of applications and most commonly in moving object detection. A very affordable way of doing so is by analysis of video sequences [1]. Currently, foreground or moving object is detected by three main methods: frame-differencing method, background subtraction method, and optical flow method. Each has their respective advantages and disadvantages [2]. However, the background subtraction method offers a more robust solution to different variations in the scene and environmental changes. The very basic idea is to generate an accurate background model that contains the stationary part of the scene and incoming frames are compared by getting the absolute difference. Then, a thresholding operation is applied to estimate the foreground object [3]. Ideally, the background is perceived to be static and foreground objects are segmented straightforwardly. Unfortunately, in practical situations, the background changes

overtime. With this in mind, an adaptive background model is used to incorporate different changes in the scene into account.

Background modeling and foreground estimation are the core components of this process. But, the accuracy and method of foreground estimation depends heavily on how the background that is modeled and its behavior, how it is initialized, and how the background is updated over time [4]. Several existing methods are proposed in this literature in recent years. Each with their offered applicability, pros and, cons. Modern background subtraction methods can be classified into one of the six categories based on how the background is modeled. Namely, statistical model, codebook based, model by sampling, rigid model, supervised machine-learning based, and hybrid model.

Statistical models are one of the most popular methods in background modeling. It is dominated by two main sub-categories, parametric and non-parametric models. Parametric models uses Gaussian Mixture Models (GMM). It models each pixel's intensity distribution by summation of weighted Gaussian distributions. Moreover, each Gaussian distributions are ordered based on the weights and variances where they are updated overtime. This method is popularized by the early works proposed by Stauffer and Grimson [5] and Zivkovic's adaption [6]. Over the years, this method became more prominent and regarded as the mainstream method in

✉ Vince Jebryl Montero  
vjm753@yahoo.com

<sup>1</sup> Department of Electronics and Communications  
Engineering, Kwangwoon University, Seoul, South Korea

background subtraction. Due to this, several methods such as [7] and [8] are proposed to improve the classical approach. An adaptive block-based scheme is proposed in [9]. While, [1, 10] and [11] proposed more innovative methods such as color invariants and texture discrimination through wavelets, respectively, to further improve accuracy based on probabilistic concept. One of the strengths of GMM is it allows multi-modal modeling. Where, multiple range of pixel value probability density can be associated in a single pixel location. As a result, it can model complex and non-static backgrounds.

The later of the two sub-categories, non-parametric statistical model uses kernel density estimation (KDE). It is based on the early works of Elgammal et al. [12] and Zivkovic's adaption [13]. Further innovation proposed in [14] which extends the concept to spatio-temporal modeling. It estimates each pixel's probability density using a kernel function from recent  $N$  samples over time. It improves two main issues in parametric models, dependency in tunable parameters and wide assumption in the models. This results to a more accurate background model that can quickly adapt to changes in the background. Since it uses  $N$  samples, this method is memory consuming. Although statistical models are flexible to different scenes, it is weak to outliers caused by noise or sudden dynamic changes.

Background models based on codebook offers feature-based models. Each pixel location is represented by codebook, compressed features (codewords) from a long image sequence (training sequence). Conceived from the early work of Kim et al. [15] to address some inherent issues in statistical models. Recent works in [16] and [17] which proposed multilayer codebook model and codeword spreading, respectively, found to be robust against dynamic and illumination changes. Codebook models are capable of modeling changes in the background over long period of time with limited memory. However, the update mechanism of codebook only updates previous codewords and does not create a new one. This can problematic if permanent structural changes are introduced in the background. In addition, the background model created from a training sequence containing clustered scenes would suffer from defective regions.

Sampling-based background models can be thought as an alternative method to statistical models. Instead of using probability density model derived on recent background values, it keeps these values for a sample consensus. These stored values are used to predict if a new pixel is either a background or foreground based on consensus mechanism. Early work from [18] uses past samples to a linear predictor and make probabilistic predictions. Recent state-of-the-art methods, [19–21], and [4] proposed background bank of multiple background models, weight samples, spatio-temporal sample consensus, and Euclidian sample consensus respectively proved to deliver more

accurate results than statistical models. But, it is likely to suffer from memory limitations and faces trade-off between latency and accuracy based on the number of stored samples.

Rigid models are the earliest and simplest conceived method of background modeling. The background model is assumed as a reference frame where the input frame is directly compared from it based on a threshold mechanism. It is initialized and updated by observing static values from frame to frame. Although simple, recent innovative works proved to produce more accurate results than classic statistical models. Proposals include entropy evaluation based on sum of absolute differences to update the model [3], a seeded block-based scheme [22] and a pseudo-random scheme of block update for UHD processing [23]. Due to its simplicity, it provides fast and memory efficient implementation as such used in [24] for surveillance video coding. However, rigid models does not offer flexibility, in which they might have low performance on special cases.

Supervised machine-learning based on deep-learning models are a recent innovation brought by advancements in computer science. As such, recent state-of-the-art works proposed in [25] based on multiscale feature triplet convolution, [26] proposed a semi-supervised multilayer of self-organizing map, [27] based on convolution with multiscale feature, and [28] based on Bayesian generative adversarial network (GAN) which dominated the top results in change detection dataset [29]. However, due to its supervised nature, such method can only be used in special applications with sufficient amount of labeled data. Not to mention the needed high-end hardware requirements for practical speed performance.

Hybrid models offer more flexibility than all mentioned categories. It combines two or more background modeling or change detection concepts that takes advantage of individual attributes. Recent works include a novel spatio-temporal flux tensor combined with split Gaussian models [30], frame differencing combined with GMM [2], and the state-of-the-art method that exploits genetic programming to select and combine many of the best existing algorithms [31]. It can result in high accuracy to different scenarios depending on the combined methods. The only disadvantage to hybrid models is the complexity and increase in compute requirements as the number of combined methods increases.

This paper presents Fast Background Subtraction with Adaptive Block Learning (FBS-ABL) using expectation value suitable for real-time performance in a wide range of platform. The following section discusses about the motivation and design basis (Sect. 2), and the details (Sect. 3) of the proposed methods. Section 4 presents the experimental results and Sect. 5 concludes the paper.

## 2 Motivation and design basis

### 2.1 Motivation

In this paper, a method for fast and “portable” background subtraction method with well-rounded performance is proposed. The portability of the design allows real-time performance in a wide range of computing platforms. Computing platforms which include high-end multi-core PCs, low-cost computers, embedded computing, accelerator assisted computing (FPGAs and GPUs), and low-cost systems. It is motivated by the fact that product and application designs are specifically engineered to meet the demands of the target market. General home surveillance may require low-cost implementations while high-end surveillance puts more computing capabilities at a higher price. It is trivial to say that low-complexity design offers economical option for price reduction. But, this leads us to the question, “why do we want a low-complexity design to run on high-end systems?” Lately, there are rapid advancements in the field of deep learning. Especially to machine vision where surveillance methods are getting smarter. The main advantage for such design is it allows fast acquisition of meaningful region proposals as input to deep-learning models. In addition, most of the computation resources can be allocated to speed-up deep-learning tasks. This eventually leads to more efficient performance in intelligent surveillance systems.

To achieve fast and well-rounded performance, this paper presents some innovations incorporated to the classical approach (rigid modeling) of background subtraction. The following are: an intermittent background update coupled with adaptation bias (compensation mechanism) and an adaptive block-based learning rate based on expected intensity difference. The following, Sect. 2.2 discusses the design basis of incorporating these innovations. While, Sects. 3.1–3.9 describe the details of the proposed method.

### 2.2 Design basis

The proposed method is based on rigid modeling in which a simple top-level structure (Fig. 1) and block-level modules following the classical approach is used. As discussed in Sect. 1, rigid models can offer efficient designs. The process flow is kept simple yet effective to avoid a lot of overheads which may affect execution time. There are certain trade-offs that must be realized to achieve high-speed execution time. Instead of utilizing multiple image channels and attributes (texture, color invariants, intensity, and pixel probability), grayscale or intensity model is used to model the background. This would simplify the probability

model over a single channel whose discrete values can only range from 0 to 255. This is apparent in the method proposed in [3] that would simplify probability distribution over a single block. Although using only the intensity model might lower the accuracy to camouflaged objects but on general scenes, fully camouflaged objects are rare. Even if part of the object is camouflaged, still significant parts can be detected. It would be enough for a surveillance camera to detect activities in the scene.

To further increase computation efficiency, the background model is not necessarily updated too often (intermittent modeling). As it would be a waste of compute resource since there will be no significant changes in the scene during few consecutive frames [22]. Background subtraction and foreground mask generation on the other hand are performed for every input frame (Fig. 1). A background update delay is introduced where the background model is updated for every  $d$  other frames as seen in Fig. 1. The value of the update delay as adapted from [22] is dependent on the scalar value of the input frame rate (fmr). The value of the update delay  $d = 2/3 \times \text{fmr}$  is found suitable based on experimental tests.

A block-based approach is used to model the background (Sect. 3.3). It decreases the number of individual parameters for evaluation and update unlike in pixel-based methods. This has the advantage of faster computing time and ignoring small noises. However, it might have a disadvantage over small background changes. An adaptive update per block that depends on the expected intensity difference between the input frame and the background model is applied (Sect. 3.5). In contrast to non-adaptive update that if there is a large intensity difference then it will also have a large effect in that block. While, a smaller intensity difference will have little effect. The adaptive update aims to equalize the effect of gaps in intensity differences which implies that a larger intensity difference will have almost the same effect with the smaller ones.

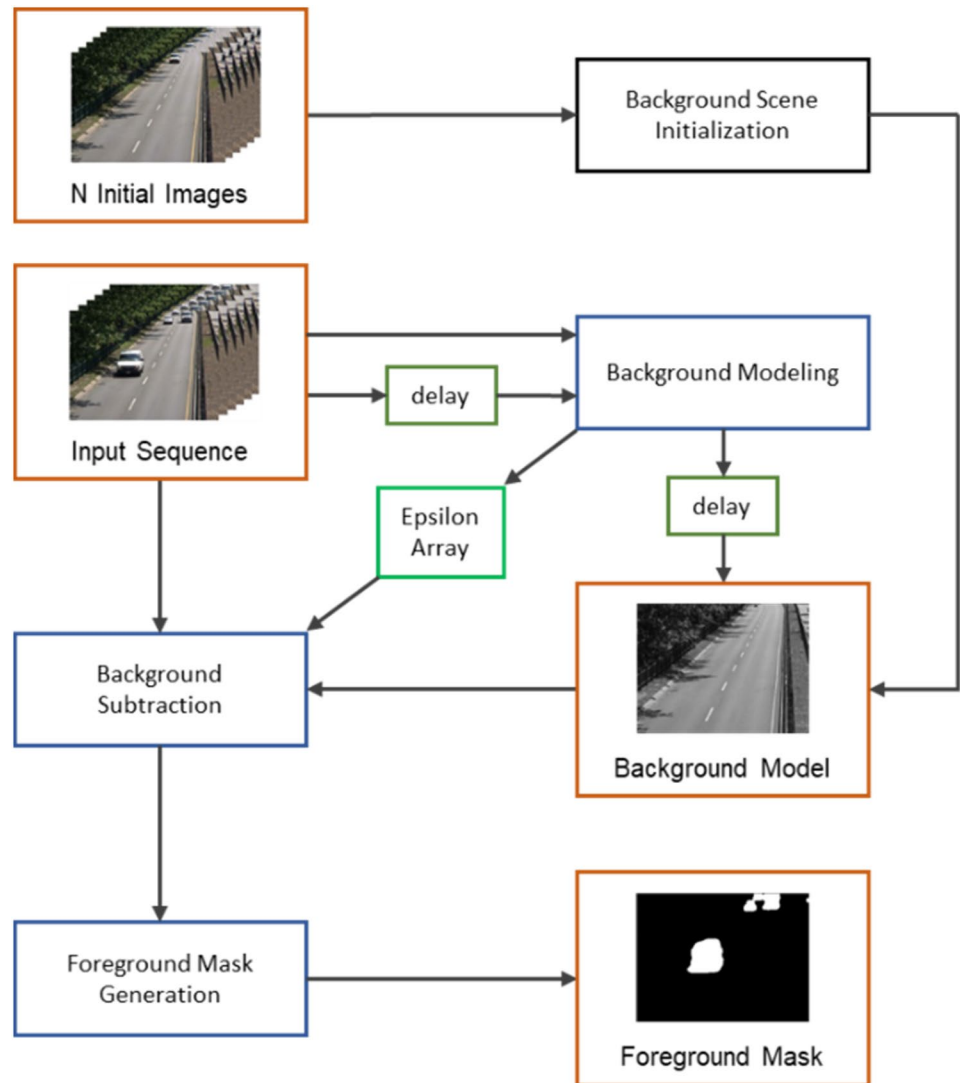
Then, a pixel-based update is applied following each block parameter to ensure smooth transition, minimizing false pixels in the locality. Slower background update is a disadvantage of the intermittent modeling which induces several false positives (adaptation ghosts) and false negatives. To compensate for such effect, an accumulative record of the past updates is kept (Delta Record). Then, an array is generated based on that record (Epsilon Array) which is subtracted from the input frame together with the background upon generating the difference image (Sects. 3.4, 3.5).

## 3 Proposed method

### 3.1 System overview

The proposed method as depicted in Fig. 1 is composed of four main modules, namely background scene initialization

**Fig. 1** Overview of the top-level flow of the proposed method



(BSI), background modeling (BGM), background subtraction (BGS), and foreground mask generation (FMG). The system takes an input from the video sequence then performs the processes to estimate the foreground object from the static background. The initial background is provided by the BSI. Then, the background model is updated only every  $d$  other frames while BGS and FMG is performed at every input frame. During background modeling, aside from updating the background model, a foreground correction factor is also generated (Epsilon Array, Sect. 3.7). The Epsilon Array is taken into account during BGS which reduces false positives while improving the foreground estimate.

### 3.2 Background scene initialization (BSI)

The background scene initialization is executed only once before the main background subtraction process begins. In practice, BSI can be done at a timely manner (hourly

and every  $n$ th hour) because the background model can be diluted with the accumulated small noises during background adaptation. The BSI takes  $N$  initial frames from the video sequence to generate the initial background model. The proposed method adapts the BSI from [3] based on the sum of absolute differences (SAD). The computational efficiency and accuracy of SAD based method is more eminent than methods based on mean and median of frames. A block-based approach is proposed in [3] to divide each input frame into  $w \times w$  blocks. Then, SAD with the next frame is computed for each block. The  $SAD^{(i,j)}(A|B)$  with  $w \times w$  pixel block inputs  $A^{(i,j)}$  and  $B^{(i,j)}$  with block coordinates  $(i, j)$  is evaluated as follows:

$$SAD^{(i,j)}(A|B) = \sum_{x=1}^w \sum_{y=1}^w |A^{(i,j)}(x, y) - B^{(i,j)}(x, y)|, \quad (1)$$

where  $(x, y)$  denotes the pixel coordinates at block  $(i, j)$  of the input image. In this context,  $A$  is the current frame ( $F_t$ ) and  $B$  is the next frame ( $F_{t+1}$ ). The value  $w = 16$  was chosen. Experimentation suggests that smaller value of  $w$  tends to accumulate noise from small objects while a larger value of  $w$  accumulates noise from larger objects. The value of  $w$  is chosen to balance the effects of both small and large objects in the scene. The initial background is then generated by combining the blocks that has the minimum value of SAD.

### 3.3 Background modeling (BGM)

The background modeling process is divided into two parts, the block evaluation, and block update with Delta Record update and Epsilon Array generation (Fig. 2). In the evaluation phase, the background image is divided into  $w \times w$  blocks where it aims to determine which blocks are to be updated. The block size  $w$  is set at 16 to balance the sensitivity from both small and large objects following the same reasoning in Sect. 3.2. During the update phase, eligible blocks are updated pixel-wise following an adaptive learning rate. The learning rate is calculated per block allowing different regions to adapt with local intensities. The Epsilon Array is generated during the adaption phase from the Delta Record; details will be explained in Sect. 3.6. Its utilization during BGS (Sect. 3.8) improves the foreground estimate as mentioned in Sect. 3.1.

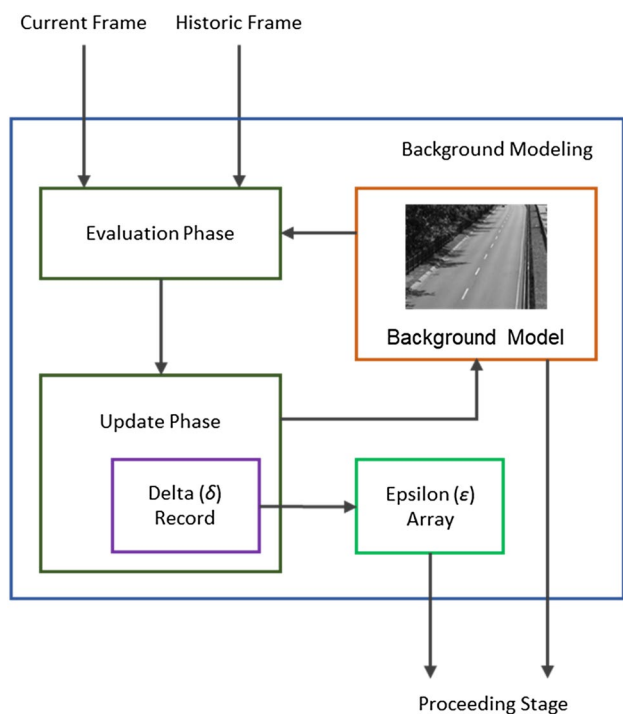


Fig. 2 Illustration of the background modeling process

### 3.4 Block evaluation phase

The evaluation process for each block is determined by the inequality described as follows:

$$\frac{|\text{SAD}^{(i,j)}(B_t|F_t) - \text{SAD}^{(i,j)}(F_{t-d}|F_t)|}{\text{SAD}^{(i,j)}(B_t|F_t)} > \rho, \tag{2}$$

where  $B_t$  is the current background,  $F_t$  is the current frame and  $F_{t-d}$  is the historic frame. The probability threshold ( $\rho$ ) is set at  $\rho = 0.50$  so that the block will neither be biased as a foreground nor assumed as background. Lowering the value of  $\rho$  increases the chance of blocks to be updated more often while a higher value results in the opposite effect. Therefore, in consequence, significantly changing the value of  $\rho$  causes sporadic false positives and negatives especially to regions with intermittent object motions and high intensity contrast. If the inequality holds true, that means a high probability of long-term change is occurring at that block (static region).

### 3.5 Block update phase

The background model is updated following the conditional statement:

$$B_{t+1}^{(i,j)} = \begin{cases} U^{(i,j)}, & \text{if value} > \rho \\ B_t^{(i,j)}, & \text{else} \end{cases}, \tag{3}$$

$$U^{(i,j)}(x, y) = (1 - \alpha^{(i,j)})B_t^{(i,j)}(x, y) + \alpha^{(i,j)}F_t^{(i,j)}(x, y), \tag{4}$$

where in Eq. (3), the “value  $> \rho$ ” indicates the block evaluation,  $B_{t+1}$  and  $B_t$  are the updated and current backgrounds, respectively, with their associated block coordinates  $(i, j)$ . Equation (4) specifies the pixel-wise update of pixels  $(x, y)$  at a certain block ( $U^{(i,j)}$ ) from  $B_t$  where  $F_t$  is the current input frame and  $\alpha^{(i,j)}$  is adaptive learning rate.

The adaptive learning rate is calculated based on the intensity difference between the blocks  $B_t^{(i,j)}$  and  $F_t^{(i,j)}$ . The formula for  $\alpha^{(i,j)}$  is described as follows:

$$\alpha^{(i,j)} = \beta \left( -m \frac{E^{(i,j)}}{2^n - 1} + c \right), \tag{5}$$

where  $n = 8$  that corresponds to 8 bit gray value,  $m$  and  $c$  are hyper-parameters with relationship  $m = c - 0.01$ . The value of  $c$  is experimentally set at  $c = 0.11$ . Tuning its value biases  $\alpha^{(i,j)}$ , increasing or decreasing the base learning rate for the background update. The set value of  $c$  allows slower background update but not too slow for midway recovery. So in effect, its value should just be right to improve true positives while lowering false positives.

The expected intensity difference ( $E^{(i,j)}$ ) and the beta ( $\beta$ ) multiplier are described in following equations:

$$E^{(i,j)} = \frac{\text{SAD}^{(i,j)}(B_t|F_t)}{w^2}, \quad (6)$$

$$\beta = \begin{cases} 8, & \text{if } \left( \sum_{i=0}^M \sum_{j=0}^N E^{(i,j)} \right) / 0.65 MN > T \\ 1, & \text{else} \end{cases}, \quad (7)$$

where in Eq. (6), the value of  $w$  is the block size equal to 16. The expected intensity difference between blocks  $B_t^{(i,j)}$  and  $F_t^{(i,j)}$  is the mean of their absolute difference with respect to the block size. The beta multiplier in Eq. (7) checks for global changes like the change in illumination. Experimentations suggest that the global change threshold  $T = 20$  is strongly suitable to detect significant changes at 65% of the total number of blocks denoted by  $MN$ . The value of  $\beta$  increases  $\alpha_{i,j}$  by eightfold when a global change is detected and accelerates the update process. Given the value of  $c$  in Eq. (5), the maximum value of  $\alpha_{i,j}$  when  $E^{(i,j)} = 0$  is 0.88. The value of 8 is chosen; it rapidly updates the block while not saturating it. If the block update is saturated, it takes longer time to adapt back in the case of intermittent illumination changes.

### 3.6 Delta Record update

The Delta Record ( $\delta$ ) holds the record of the accumulated absolute value of every pixel-wise update in each block. It is used in generating the Epsilon Array ( $\epsilon$ ) in which during BGS will induce an adaptation bias. This will cause moving objects that stopped moving to adapt slower in the background while adaptation will be faster for objects that remained static then starts to move again (Fig. 3). This improves the accuracy

of the system. Updating of the delta record is only done on blocks which the evaluation process holds true (“value  $> \rho$ ”). The update of the Delta Record is described as follows:

$$D^{(i,j)}(x, y) = \alpha^{(i,j)} \left( B_t^{(i,j)}(x, y) - F_t^{(i,j)}(x, y) \right), \quad (8)$$

$$\delta_{t+1}^{(i,j)}(x, y) = \begin{cases} \delta_t^{(i,j)}(x, y) + |D^{(i,j)}(x, y)|, & \text{if } E^{(i,j)} > \tau \\ 0, & \text{else} \end{cases}, \quad (9)$$

where the value of  $\delta_{t+1}^{(i,j)}(x, y)$  is reset to 0 at every pixel ( $x, y$ ) whenever  $E^{(i,j)}$  falls below the hard threshold ( $\tau$ ) else it is updated. The value of the threshold is the same with hard threshold used at FMG described in Sect. 3.9. The value of  $\delta_{t+1}^{(i,j)}(x, y)$  is also reset to 0 on these conditions, when its value is greater than  $2^8 - 1$  and whenever the conditions for global change holds true ( $\beta = 8$ ).

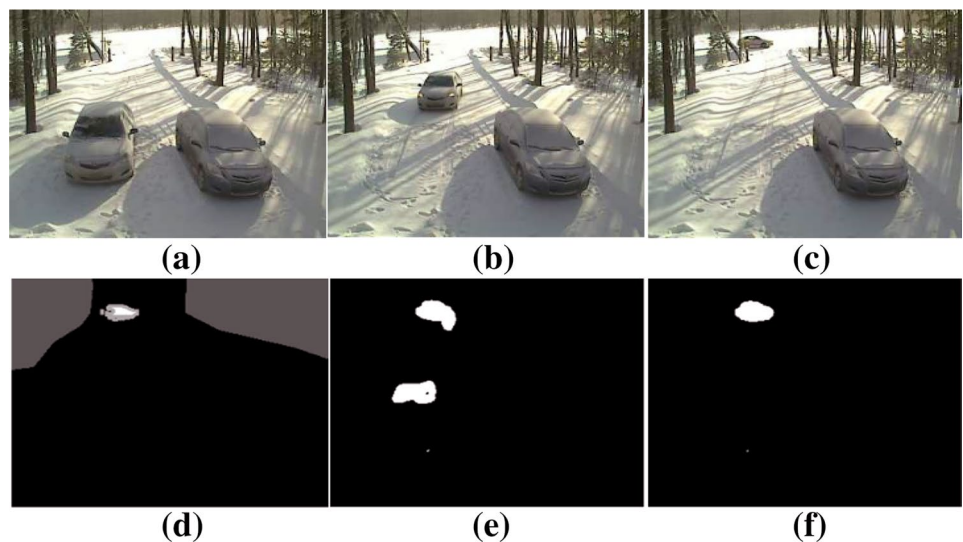
### 3.7 Epsilon Array generation

The Epsilon Array counteracts the effect of slow background update by acting as a counter weight providing adaptation bias during BGS (Sect. 3.8). In [9], an adaptive learning rate based on block neighborhood is proposed to induce the same adaptation bias to counter changes in motion state. But, this is a more expensive method because each block also needs to keep track of neighboring blocks. To generate the Epsilon Array ( $\epsilon$ ) from  $\delta$ , the relationship is described as follows:

$$\mu_0^{(i,j)} = u \left( \frac{E^{(i,j)}}{2^n - 1} \right) + v, \quad (10)$$

$$\mu^{(i,j)} = \begin{cases} 1, & \text{if } \mu_0^{(i,j)} > 1 \\ \mu_0^{(i,j)}, & \text{else} \end{cases}, \quad (11)$$

**Fig. 3** Comparison between background-to-foreground adaptation without adaptation bias and with adaptation bias in winter driveway, [29]. Initially, the car on the left is static for a long time, then it started moving and maneuvered as depicted above. **a** Initial position of the car (left). **b** Car on the left transition. **c** Final position of the car (current frame). **d** Ground truth on image in c. **e** Output without adaptation bias. **f** Output with adaptation bias



$$\varepsilon^{(i,j)} = \mu^{(i,j)} \delta_{t+1}^{(i,j)}, \quad (12)$$

where in Eq. (10),  $n = 8$  represents gray value and the slope  $u = 1.275$  which is derived from the  $x$  coordinate cutoff point of  $2/3 \times (2^8 - 1)$  at  $\mu_0 = 1$  and the hyper-parameter  $v = 0.15$ . The value of  $v$  biases the adaptive weight  $\mu^{(i,j)}$ , therefore, controlling the amount of counteractive effect of  $\varepsilon$  on the output. Drastic change in the value of  $v$  may lead to poor suppression of false negatives. On the other hand,  $\mu^{(i,j)}$  is the stabilizing factor of  $\varepsilon^{(i,j)}$  as  $\delta_{t+1}^{(i,j)}$  accumulates every update.

### 3.8 Background subtraction (BGS)

The background subtraction process aims to estimate the foreground object from the input frame by subtracting the background model. Ideally, this can be done with a simple subtraction operation. But, because of factors such as dynamic motion in the background, white noise, adaptation noise, and other sources of noise, different methods have been proposed to estimate the foreground [1–3, 5, 7–9, 11, 14, 16, 18, 24–26, 30]. In the proposed method, background subtraction process is applied to generate the difference image ( $S$ ) as follows:

$$S(x, y) = |a_k * F_t(x, y) - \varepsilon(x, y) - a_k * B_t(x, y)|, \quad (13)$$

where  $a_k * I(x, y)$  is the convolution between the image and the average blur kernel  $a_k$  with kernel size  $k$ . Appropriate range of kernel size  $k : 3 \leq k \leq 9$ ; in which  $k$  can also be regarded as a dynamic sensitivity parameter. The average blur acts as a low-pass filter to smoothen out the noisy structure of each image. In addition, it is chosen among other filters because of the simplicity of its operations allowing fast executions.

### 3.9 Foreground mask generation (FMG)

The foreground mask generation aims to transform the difference image  $S(x, y)$  into a binary image that labels the foreground and the background (Fig. 4). In doing so, after BGS,  $S(x, y)$  is filtered by average blur to flatten out regions containing narrow depressions and ridges. This will effectively filter out granular noise and connect potentially separated blobs during the thresholding process. The thresholding operation is done by implementing a hard threshold  $\tau$ . The values from the filtered difference image  $a_g * S(x, y)$  below this threshold is labeled as background (0), otherwise a foreground (1). The suitable range of values of  $\tau$  is determined to be around 10% of  $2^8$  gray levels from empirical testing. In which  $\tau$  can also be viewed as a contrast sensitivity parameter to detect small changes in the foreground.

## 4 Experimental results

In this section, experimentation and performance evaluation is done to determine the relevance of our proposed method. The evaluation for quantitative and executional performance is done to test the accuracy and running time. In the quantitative evaluation, our method is compared on both widely popular and state-of-the-art algorithms by conducting experimentation on a widely used dataset. While in executional evaluation, both custom and widely used dataset are tested on different platforms to compare its execution speed with other proposed methods.

### 4.1 Quantitative method

A widely used dataset for the quantitative measure of performance of background subtraction algorithms is the Change-Detection.net 2014 (CDnet 2014) dataset [32]. In this dataset, 4–6 sequences are presented for 11 different categories to test the accuracy of FBS-ABL. To provide a valid measure of the result, three evaluation metrics are used, namely, Recall (Re), Precision (Pr), and F-measure (Fm). These metrics rely on the pixel count basis namely, true positive (TP), false positive (FP), true negative (TN), and false negative (FN) that are generated using the provided ground truth data.

Recall is a metric to measure the completeness of the detected foreground, in which the value goes higher when the rate of false negatives is low. Precision on the other hand, is the measure of exactness of the detected foreground, in which the value will decrease when the rate of false positives is high. F-measure is used to measure the balance of Recall and Precision with equal weights which implies that it is high only when both Precision and Recall are high. The formula for the F-measure can be described as follows:

$$Fm = (2 \times Pr \times Re) / (Pr + Re), \quad (14)$$

where  $Re = TP / (TP + FN)$  and  $Pr = TP / (TP + FP)$  are both described by the pixel count basis [32, 33]. These metrics were chosen from a pool of different metrics because they present non-redundancy in the quantitative and semantic interpretation of the data.

In comparing between the results from various existing methods published in the CDnet website [29], the median value from each category (CDnet-14) will be used as the baseline. This is done to avoid a biased comparison. Using the mean would drastically pull down the results and using the top percentile only would fail to represent the widely accepted methods. To compare with older but popular methods, GMM [5], KDE [12], GMM2 [6], and KNN [13] were chosen. ViBe [4] is also presented which is one of the fastest algorithms available but only on the CDnet2012 [33] dataset

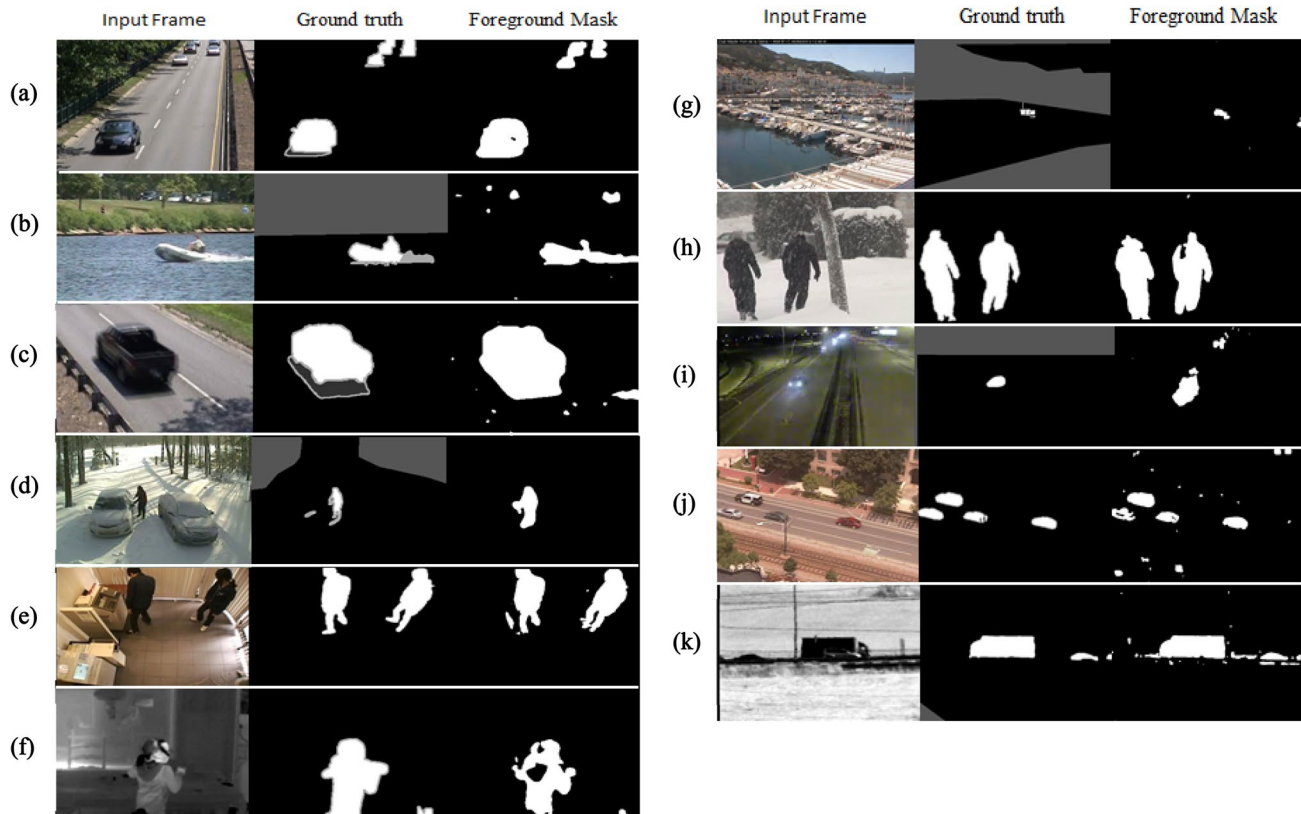
(first six categories of CDnet2014 [32]) because of data availability online. In addition, five recent, state-of-the-art methods were chosen, [19–21, 30], and [31] for comparison which presented a complete evaluation for CDnet2014. For comparison with block-based methods, the average Fm of [3] and [9] is presented for the first eight categories of CDnet 2014 because of data availability and the concepts are much more relatable to the proposed method.

## 4.2 Quantitative performance

In Tables 1 and 2, the best performing methods for each category based on Fm is highlighted in italics. It can be seen that it is dominated by the five state-of-the-art methods [19–21, 30] [31], as expected but with low execution speed which is not suitable for real-time applications. Most state-of-the-art methods are more focused on the performance rather than applicability. The classical methods such as [5, 6, 12, 13] and fast methods such as ViBe [4] are more suitable for real-time applications which can be easily adapted for different platforms. The trade-off between performance and execution speed can be seen in Table 1, and as it shows, widely adaptable methods have fast/practical execution at the cost of reduced performance. But,

the overall performance of FBS-ABL in Table 2 shows much better score than the classical methods with superior execution speed as shown in Table 1 and 3, 4 and 5 for different platforms. In performance comparison with state-of-the-art methods where the focus is to perform well on most categories, FBS-ABL may only be comparable only on certain categories such as Dynamic Background, Night Videos, Intermittent Motion, and Bad Weather. This is can be explained based on the design trade-off which will be detailed on the succeeding paragraphs.

The overall score in Table 2 was calculated by averaging the performance from the eleven categories of CDnet2014. It is expected that the five state-of-the-art methods will have the top rankings according to Fm. FBS-ABL falls just nearly below CDnet-14 (the median of various existing methods in the CDnet website [29]) in terms of Fm ranking. Nearly half of the methods of CDnet-12 includes improved and state-of-the-art performance methods while the other half are classical methods that are commonly adapted to wide applications. The lower average Fm score of FBS-ABL compared to CDnet-14 is due to the fact that it has a very high average Re, higher than most of the state-of-the-art methods (2nd rank) while its average Pr is comparably lower (8th rank).



**Fig. 4** Foreground results obtained from each category of CDnet 2014 dataset [32]. **a** Baseline. **b** Dynamic background. **c** Camera Jitter. **d** Intermittent object motion. **e** Shadows. **f** Thermal. **g** Low frame rate. **h** Bad weather, **i** Night videos. **j** PTZ. **k** Turbulence



**Table 1** Comparison on the first six categories of CDnet 2014 dataset

Category	Metrics	CDnet-14 [29]	GMM [5]	GMM2 [6]	KDE [12]	KNN [13]	ViBe [4]	FTSG [30]	SubS [21]	WeSB [20]	MBS [19]	IUTIS [31]	FBS-ABL
Baseline	<b>Re</b>	0.9158	0.8180	0.8085	0.8969	0.7934	0.8204	0.9513	0.9520	0.9422	0.9158	0.9680	<b>0.8910</b>
	<b>Pr</b>	0.9223	0.8461	0.8993	0.9223	0.9245	0.9288	0.9170	0.9495	0.9422	0.9431	0.9464	<b>0.8602</b>
	<b>Fm</b>	0.9192	0.8245	0.8382	0.9092	0.8411	0.8700	0.9330	0.9503	0.9413	0.9287	<b>0.9567</b>	<b>0.8649</b>
Dynamic back-ground	<b>Re</b>	0.8016	0.8344	0.8019	0.8012	0.8047	0.7222	0.8691	0.7768	0.6796	0.7641	0.8636	<b>0.7958</b>
	<b>Pr</b>	0.7607	0.5989	0.6213	0.5732	0.6931	0.5346	0.9129	0.8915	0.8933	0.8651	0.9324	<b>0.7332</b>
	<b>Fm</b>	0.7393	0.6330	0.6328	0.5961	0.6865	0.5652	0.8792	0.8177	0.7440	0.7915	<b>0.8902</b>	<b>0.7424</b>
Camera jitter	<b>Re</b>	0.7437	0.7334	0.6900	0.7375	0.7351	0.7375	0.7717	0.8243	0.7777	0.8321	0.8220	<b>0.8046</b>
	<b>Pr</b>	0.8021	0.5126	0.4872	0.4862	0.7018	0.4862	0.7645	0.8115	0.8395	0.8443	0.8511	<b>0.4656</b>
	<b>Fm</b>	0.7207	0.5969	0.5670	0.5720	0.6894	0.5720	0.7513	0.8152	0.7976	<b>0.8367</b>	0.8332	<b>0.5298</b>
Intermittent object motion	<b>Re</b>	0.6578	0.5142	0.5467	0.5035	0.4617	0.5122	0.7813	0.6578	0.7472	0.7418	0.7047	<b>0.7861</b>
	<b>Pr</b>	0.7374	0.6688	0.6458	0.4609	0.7121	0.6515	0.8512	0.7957	0.7888	0.7827	0.8501	<b>0.7943</b>
	<b>Fm</b>	0.5810	0.5207	0.5325	0.4088	0.5026	0.5074	<b>0.7891</b>	0.6569	0.7392	0.7568	0.7296	<b>0.7232</b>
Shadow	<b>Re</b>	0.8543	0.7960	0.7774	0.8541	0.7478	0.7833	0.9214	0.9419	0.9401	0.7920	0.9492	<b>0.9143</b>
	<b>Pr</b>	0.8291	0.7156	0.7232	0.7660	0.7788	0.8342	0.8535	0.8646	0.8686	0.8262	0.8766	<b>0.8569</b>
	<b>Fm</b>	0.8128	0.7370	0.7322	0.8030	0.7468	0.8032	0.8832	0.8986	0.8999	0.7968	<b>0.9084</b>	<b>0.8671</b>
Thermal	<b>Re</b>	0.6929	0.5691	0.5542	0.6725	0.4817	0.5435	0.7357	0.8161	0.7727	0.8162	0.7990	<b>0.6394</b>
	<b>Pr</b>	0.8715	0.8652	0.8706	0.8974	0.9186	0.9363	0.9088	0.8328	0.8554	0.8268	0.8969	<b>0.8002</b>
	<b>Fm</b>	0.7597	0.6621	0.6548	0.7423	0.6046	0.6647	0.7768	0.8171	0.7962	0.8194	<b>0.8303</b>	<b>0.6619</b>
Compute platform		Core i7 3.4 GHz 720×480	Core i7 3.4 GHz 720×480	Core i7 3.4 GHz 720×480	Core i7 3.4 GHz 720×480	Core i7 3.4 GHz 720×480	Core i7 2.67 GHz 640×480	Laptop Core 2 Duo 320×240	PC-i5 3.3 GHz 320×240	Core i5 3.2 GHz 320×240	Core i5 PC 320×240	PC Env 320×240	Core i7 3.3 GHz 720×480
Video resolution		720×480	720×480	720×480	720×480	720×480	640×480	320×240	320×240	320×240	320×240	320×240	720×480
Execution speed (fps)		21	49	9	9	25	200	10	45	2	9	10	217

The CDnet-14 column presents the median values from results in [29]. It is not an independent method, therefore, execution speed does not apply. Data for the proposed method FBS-ABL is highlighted in bold

**Table 2** Comparison on the newer categories of CDnet 2014 dataset

Category	Metrics	CDnet-14 [29]	GMM [5]	GMM2 [6]	KDE [12]	KNN [13]	FTSG [30]	SubS [21]	WeSB [20]	MBS [19]	IUTIS [31]	FBS-ABL
Low frame rate	<b>Re</b>	0.7158	0.5823	0.5300	0.7000	0.6290	0.7517	0.8537	0.8842	0.6773	0.8398	<b>0.6616</b>
	<b>Pr</b>	0.6402	0.6894	0.6686	0.6245	0.6865	0.6550	0.6035	0.6134	0.6000	0.7424	<b>0.7313</b>
	<b>Fm</b>	0.6068	0.5373	0.5065	0.5478	0.5491	0.6259	0.6445	0.6602	0.6350	<b>0.7743</b>	<b>0.6328</b>
Bad weather	<b>Re</b>	0.7181	0.7181	0.6863	0.6941	0.6537	0.7457	0.8273	0.8168	0.8341	0.7493	<b>0.7449</b>
	<b>Pr</b>	0.8718	0.7704	0.8138	0.8486	0.9114	0.9231	0.9091	0.9134	0.7828	0.9311	<b>0.8965</b>
	<b>Fm</b>	0.7730	0.7380	0.7406	0.7571	0.7587	0.8228	<b>0.8619</b>	0.8606	0.7980	0.8248	<b>0.8106</b>
Night videos	<b>Re</b>	0.5852	0.5261	0.4797	0.5914	0.5413	0.6107	0.6570	0.6370	0.5535	0.5852	<b>0.7498</b>
	<b>Pr</b>	0.5021	0.4128	0.4231	0.4036	0.4298	0.4904	0.5359	0.5827	0.4899	0.5438	<b>0.4957</b>
	<b>Fm</b>	0.4946	0.4097	0.3960	0.4365	0.4200	0.5130	0.5599	<b>0.5929</b>	0.5158	0.5290	<b>0.5272</b>
PTZ	<b>Re</b>	0.6863	0.6475	0.6111	0.8121	0.6980	0.6730	0.8306	0.8145	0.5973	0.6749	<b>0.8357</b>
	<b>Pr</b>	0.2793	0.1185	0.0683	0.0188	0.1979	0.2861	0.2840	0.3121	0.5400	0.3833	<b>0.2290</b>
	<b>Fm</b>	0.3176	0.1522	0.1046	0.0365	0.2126	0.3241	0.3476	0.3844	<b>0.5520</b>	0.4282	<b>0.3267</b>
Turbulence	<b>Re</b>	0.7218	0.7913	0.7786	0.8492	0.7682	0.6109	0.8050	0.7382	0.6037	0.6777	<b>0.9468</b>
	<b>Pr</b>	0.6809	0.4293	0.3494	0.3908	0.5117	0.9035	0.7814	0.8371	0.6198	0.9414	<b>0.4936</b>
	<b>Fm</b>	0.6275	0.4663	0.4169	0.4478	0.5198	0.7127	<b>0.7792</b>	0.7737	0.5858	0.7836	<b>0.5564</b>
CDnet 2014 average	<b>Re</b>	0.7358	0.6846	0.6604	0.7375	0.6650	0.7657	0.8124	0.7955	0.7389	0.7849	<b>0.7973</b>
	<b>Pr</b>	0.7179	0.6025	0.5973	0.5811	0.6787	0.7696	0.7509	0.7679	0.7382	0.8087	<b>0.6688</b>
	<b>Fm</b>	0.6684	0.5707	0.5566	0.5688	0.5937	0.7283	0.7408	0.7445	0.7288	<b>0.7717</b>	<b>0.6585</b>

Data for the proposed method FBS-ABL is highlighted in bold

**Table 3** Comparison with block-based methods

Metric	Reference [3]	Reference [9]	FBS-ABL
CDnet 2014—8c (average Fm)	0.7246	0.6811	<b>0.7290</b>
Platform	Laptop-i3, 4 GB RAM	Laptop-i3, 4 GB RAM	<b>Laptop-i3, 4 GB RAM</b>
Resolution	320×256	320×256	<b>640×480</b>
Execution (fps)	22	17	<b>155</b>

CDnet 2014—8c: the first eight categories of CDnet 2014

Data for the proposed method FBS-ABL is highlighted in bold

**Table 4** Speed comparison on other methods and platforms

Method	Platform	Resolution	fps
MCBS [16]	PC w/NVIDIA GTX 970	320×240	18
Reference [8]	Intel Core-2 2.26 GHz	320×240	20
SMSOM-BM [26]	PC w/NVIDIA GTX 260	320×240	32.3
GMM2 [6]	Laptop-i3, 4 GB RAM	320×240	126
KNN [13]	Laptop-i3, 4 GB RAM	320×240	82
FBS-ABL	<b>Laptop-i3, 4 GB RAM</b>	<b>320×240</b>	<b>435</b>
FBS-ABL	<b>PC-i7 3.3 GHz, 16 GB RAM</b>	<b>320×240</b>	<b>655</b>
ABPBGs [22]	PC-i7 2.8 GHz, 16 GB RAM	360×270	36
MBBM [23]	PC-i7 2.8 GHz, 16 GB RAM	360×270	46
FWFC [11]	PC-i7 4790, 32 GB RAM	640×384	3.64
GMM2 [6]	Laptop-i3, 4 GB RAM	640×480	56
KNN [13]	Laptop-i3, 4 GB RAM	640×480	30
ViBe (RGB) [4]	PC-i7 2.67 GHz 6 GB RAM	640×480	200
ViBe (gray) [4]	PC-i7 2.67 GHz 6 GB RAM	640×480	250
FBS-ABL	<b>Laptop-i3, 4 GB RAM</b>	<b>640×480</b>	<b>155</b>
FBS-ABL	<b>PC-i7 3.3 GHz, 16 GB RAM</b>	<b>640×480</b>	<b>277</b>
ABPBGs [22]	PC-i7 2.8 GHz, 16 GB RAM	4 K UHD	5.23
MBBM [23]	PC-i7 2.8 GHz, 16 GB RAM	4 K UHD	4.51
FBS-ABL	<b>PC-i7 3.3 GHz, 16 GB RAM</b>	<b>4 K UHD</b>	<b>15</b>

Data for the proposed method FBS-ABL is highlighted in bold

**Table 5** Speed on software implementation in embedded system

Method	Platform	Resolution	fps
Reference [1]	Raspberry Pi	320×240	3.10
ViBe (scaled) [4]	Embedded ARM Core	320×240	6
MBSCIGA [10]	ARM Cortex-A9 2-Core	320×240	15.2
FBS-ABL	<b>ARM Cortex-A9 2-Core</b>	<b>320×240</b>	<b>83</b>

Data for the proposed method FBS-ABL is highlighted in bold

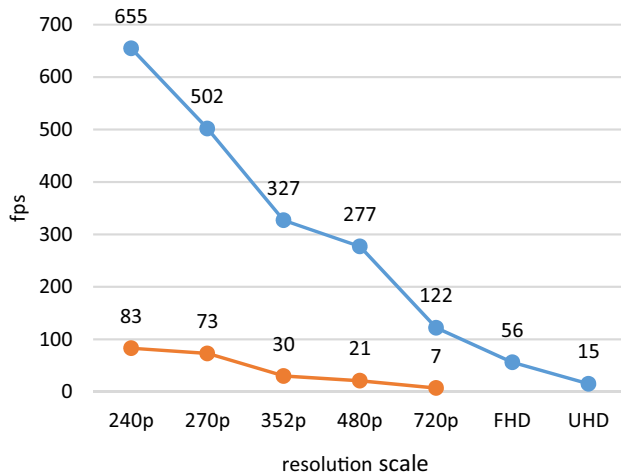
The high Re and lower Pr of FBS-ABL in the overall score is because it is weak to the cases such as Camera Jitter and Turbulence which dragged down the average Pr. In these cases, the background is constantly changing and because

of the intermittent modeling scheme discussed in Sect. 2.2 the background update cannot keep up with abrupt changes where the update is only once every few frames. In addition, the block-based method causes the false-positive regions to be thicker (see Fig. 6). But in general, these cases are not frequently encountered on most applications. However, a better performance can be seen on Dynamic Background because the changes are more localized which can be suppressed by the block-based update and low-pass filter in the foreground mask generation stage. FBS-ABL also performed satisfactorily on the Baseline category relative to CDnet-14 which is within 6% error difference and is relatively comparable to the widely adapted methods.

In categories like thermal, where camouflage effect is most likely due to the nature of thermal image acquisition. FBS-ABL has a poor Fm score than most of the presented method. The camouflage effect is prominent as a consequence of using only the intensity scale which causes the Re to be low. In bad weather, FBS-ABL has a comparable Fm with some of the state-of-the-art methods because the block-based method is insensitive to small background noises such as rain and snow. Both Night Videos and PTZ also presents scenes with global changes like sudden change in illumination and change of background scene where FBS-ABL can adapt satisfactorily given its intermittent updating scheme. In which, resulted in comparable results with two to three state-of-the-art methods. Overall, FBS-ABL shows practical and well-rounded performance especially to conditions that are frequently encountered in different application and in general scenes such as Baseline, Dynamic Background, Night Videos, Intermittent Motion, and Bad Weather.

### 4.3 Executorial performance

In executorial performance, FBS-ABL is tested on different platforms including embedded platform which can be seen in the Tables 1, 3, 4, and 5. Sequences from the CDnet 2014 [32] containing resolution of 320×240 pixels are used to evaluate the said scale while a custom Baseline-like dataset is used to evaluate higher resolutions up to high definition (HD) scales. Based on the top-level structure, the execution



**Fig. 5** Average execution speed of FBS-ABL in different resolution scales in PC and embedded platform: PC-i7@3.3 GHz with 16 GB RAM (blue) and ARM Cortex-A9 2-Core, 1 GB RAM (orange). (240p: 320×240, 270p: 360×270, 352p: 640×352, 480p: 640×480, 720p: 1280×720, FHD: 1920×1080, UHD: 3840×2160)

**Table 6** Comparison of execution time of FBS-ABL with and without FPGA accelerator using DE1-SoC Development Board

Platform	Image resolution	fps
ARM cortex only	640×480	21
ARM with low cost FPGA	640×480	40

speed of FBS-ABL does not greatly vary from different scenes in the same resolution scales. The execution time variation is found to be within  $\pm 10\%$  in higher resolutions and within  $\pm 15\%$  in lower resolutions.

In Tables 1, 3, 4, and 5, the execution speed of FBS-ABL is compared to other methods in different resolution scales on different platforms. It can be observed that FBS-ABL dominates in speed performance in all platforms. In Table 3, although FBS-ABL has comparable results with [3] but shows better execution speed even at a higher resolution. This shows the effective design strategy behind FBS-ABL even when compared to other block-based methods. The execution time per resolution scale in the embedded platform is shown in Fig. 5 and achieved a peak execution speed of 83 fps at 240p resolution. In Table 6, the execution time per resolution scale in PC environment is shown and the trend in execution time is similar to that in embedded platform. Implementation in a PC environment for 720p to UHD scale is done with core optimizations that takes advantage of the processor architecture with multiple logical processors to parallelized

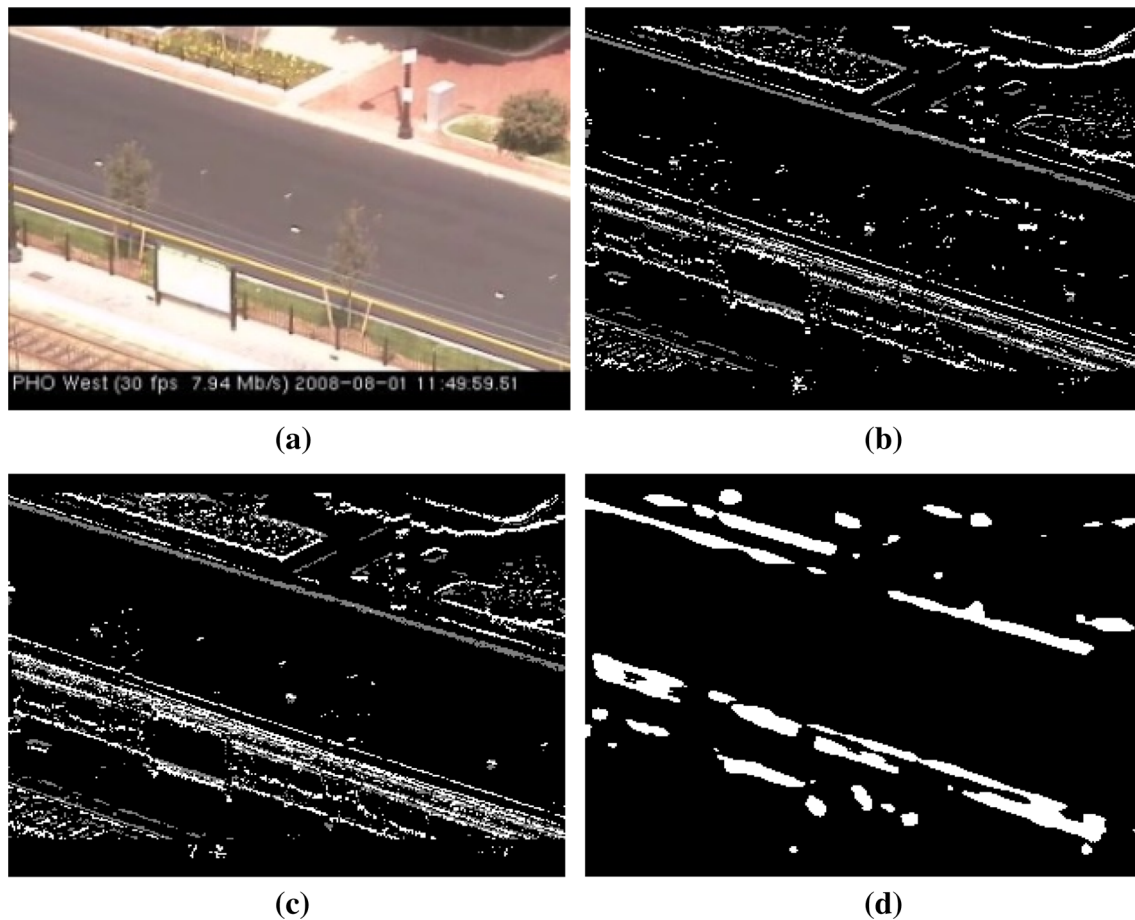
local operations. In Table 6, it is shown that FBS-ABL can greatly benefit from a compute accelerator such as the FPGA. Implementation for CPU-FPGA embedded co-design is done with OpenCL using the DE1-SoC development board which houses the ARM Cortex-A9 processor together with the Cyclone V low-cost FPGA.

#### 4.4 Comparison with other methods

The performance of FBS-ABL compared to conventional methods [5, 6, 12, 13] and newer state-of-the-art methods [19–21, 30, 31] are shown in Tables 1 and 2. It can be seen that the performance of FBS-ABL is generally higher than convention methods for most of the categories except in the Camera Jitter category where the background is constantly moving. It should be noted that most of these methods are pixel based while FBS-ABL update scheme is block-based and also the background is only updated once every few frames. These means that a whole block contributes to every regional update on the background. Since the background in the current frame is moving constantly in the case of camera jitter, a thicker false positive is created in that region. In addition, these false-positive regions can be amplified by the low-pass filter during foreground mask generation. In Fig. 6, the camera jitter behavior for FBS-ABL is shown in comparison to two other conventional pixel-based methods. It can be immediately observed that false positives for FBS-ABL are more concentrated in a number of regions while for the two other methods are much more scattered.

The performance of FBS-ABL does not greatly deviate in most of the presented categories such as Baseline, Dynamic Background, Intermittent Motion Objects, Shadows, Night Videos, and PTZ compared to the state-of-the-art methods. In the categories Bad Weather and Low Frame rate, FBS-ABL shows more robustness than most of the methods. However, in execution speed the grayscale version of ViBe is the only method it can be on par with while besting all other methods. Speed up version of conventional methods using a method of attention sampling in [34] allows a faster execution in Full HD resolution sequence. With reference to the paper [34], GMM [5], KDE [12], and GMM2 [6] execution speeds perform at 18.6 fps, 31.5 fps, and 29.7 fps, respectively, running only at a single core in PC environment. For comparison, FBS-ABL is also executed using single-core implementation which performed at 32 fps in Full HD sequence, slower than its multicore implementation at 56 fps shown in Fig. 5. While FBS-ABL are on par with the speed-up versions of conventional methods, Table 1 shows that FBS-ABL performs better.

In high-end, very high resolution of 4 K UHD, FBS-ABL is compared to two other methods, ABPBGs [22] and MBBM [23] which are designed for UHD background subtraction (Table 4). It can be observed that FBS-ABL can



**Fig. 6** Comparison on the behavior for Camera Jitter category on the 380th frame of boulevard sequence [29]. **a** Current input frame. **b** GMM2 [6]. **c** KNN [13]. **d** FBS-ABL (proposed method)

perform  $5 \times$  faster but with much greater memory usage. In comparison regarding the memory usage, in 4 K UHD FBS-ABL requires more than 5 GB while ABPBGs and MBBM uses only lower than 450 MB as claimed in their paper. However, it is possible to lower the memory requirements of FBS-ABL by removing pre-stored states but at the cost of processing speed.

In Table 5, execution speed of software implementation in embedded system is shown for FBS-ABL, downscaled ViBe [4], MBSCIGA [10] and method in [1]. A lower execution speed is expected in [1] because of its inherent complexity. However, hardware-oriented version with the same algorithm concept in [1] is proposed by the same authors in MBSCIGA. Its software implementation achieved  $5 \times$  faster speed than its predecessor but FBS-ABL outperforms it by  $5 \times$  the speed. Although, MBSCIGA performs short compared to FBS-ABL in software implementation but its strength comes in hardware. As claimed by their paper [10], it can perform up to 74 fps with FHD resolution in full hardware implementation. As it is right now, FBS-ABL is software centered. A full hardware implementation means

that some functions must be tweaked to achieve comparable results with MBSCIGA. This software centeredness is also apparent in Table 6 where overheads from the FPGA-ARM memory access slows down the speed. It only achieved  $2 \times$  speed up from a low-cost FPGA.

With reference to paper [4], the downscaled version of ViBe can run at more than 350 fps for  $640 \times 480$  sequence in PC which surpasses FBS-ABL for a performance cost of 5% degradation. A main contributing factor about the low execution speed of downscaled ViBe in embedded system is the slower memory access compared to PC. Downscaled ViBe uses  $N=5$  samples plus the current frame totals  $6 \times$  resolution  $\times$  pixel byte memory and also the search speed of 5 samples per current pixel per generation of the mask in comparison to FBS-ABL which only needs to access  $3 \times$  resolution  $\times$  pixel byte per mask generation, only half as much as downscaled ViBe. In addition, not to mention that FBS-ABL requires no pixel search to generate the mask which makes it more suitable for real-time performance in embedded system.

## 5 Conclusion

This paper proposes a Fast Background Subtraction with Adaptive Block Learning using expectation value (FBS-ABL) that is suitable for real-time performance in wide range of platforms. The background model updating is based on the intermittent approach coupled with adaptive block-learning rate, and a devised method for background adaptation bias. An efficient background subtraction method is used in the segmentation process. FBS-ABL achieved a well-rounded performance in varying scenes with overall results better than conventional techniques but outperforms them in execution speed. Its simplistic approach together with efficient design allows it to execute at high speeds in a wide range of platform.

The proposed method, FBS-ABL achieved a fast execution speed of up to 56 fps in PC using Full HD video. It also achieved 655 fps and 83 fps in PC and ARM core-embedded platform, respectively, using the minimum input resolution of  $320 \times 240$ . The efficient but practical approach of FBS-ABL demonstrated its adaptability for both low-quality and high-quality image processing with the potential to parallelize computation for high resolutions to execute at real-time speeds. Although it is particularly weaker to cases like camera jitter, but it does not occur frequently on general scenes. Future modifications, that include a threshold adaptation process per block that will remedy this inherent weakness but for a trade-off of speed. Its fast execution speed has also the advantage of more room for post processes including deep-learning tasks such as object recognition, and motion tracking. Overall, it can be deduced that FBS-ABL is effective performance-wise and is suitable for real-time performance applications.

**Acknowledgements** This work was supported by Kwangwoon University and by the MISP Korea under the National Program for Excellence in SW (2017-0-00096) supervised by IITP.

## References

- Cocorullo, G., Frustaci, F., Guachi, L., Perri, S.: Embedded surveillance system using background subtraction and raspberry Pi. In: 2015 AEIT International Annual Conference (AEIT), pp. 1–5 (2015)
- Xu, Z.X., Zhang, D.H., Du, L.: Moving object detection based on improved three frame difference and background subtraction. In: International Conference on Industrial Informatics—Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), pp. 79–82 (2017)
- Elharrouss, O., Abbad, A., Moujahid, D., Tairi, H.: Moving object detection zone using a block-based background model. *IET Comput. Vis.* **12**(1), 86–94 (2018)
- Barnich, O., Droogenbroeck, M.V.: ViBe: a universal background subtraction algorithm for video sequences. *IEEE Trans. Image Process.* **20**(6), 1709 (2011)
- Stauffer, C., Grimson, W.E.L.: Adaptive background mixture for real-time tracking. In: Proceedings on IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2 (1999)
- Zivkovic, Z.: Improved adaptive Gaussian mixture model for background subtraction. In: Proceedings of the 17th international conference on pattern recognition, Vol. 2, pp. 28–31 (2004)
- Riahi, D., St. Onge, P.L., Bilodeau, G.A.: RECTGAUSS-TeX: block-based background subtraction. École Polytechnique de Montréal, Montréal (2012)
- Shah, M., Deng, J.D., Woodford, B.J.: Improving mixture of Gaussians background model through adaptive learning and spatio-temporal voting. In: IEEE International Conference on Image Processing, pp. 3436–3440 (2013)
- Lin, D.Z., Cao, D.L., Zeng, H.L.: Improving motion state change object detection by using block background context. In: 14th UK Workshop on Computational Intelligence (UKCI), pp. 1–6 (2014)
- Cocorullo, G., Corsonello, P., Frustaci, F., Guachi, L.A.G., Perri, S.: Multimodal background subtraction for high-performance embedded systems. *J. Real Time Image Process.* (2016). <https://doi.org/10.1007/s11554-016-0651-6>
- Li, S., Florencio, D., Li, W., Zhao, Y., Cook, C.: A fusion framework for camouflaged moving foreground in the wavelet domain. *IEEE Trans. Image Process.* **27**(8), 3910–3930 (2018)
- Elgammal, A., Harwood, D., Davis, L.: Non-parametric model for background subtraction. In: Proceedings of the European Conference on Computer Vision, Lectures Notes in Computer Science, Vol. 1843, pp. 751–767 (2000)
- Zivkovic, Z., van der Heijden, F.: Efficient adaptive density estimation per image pixel for the task of background subtraction. *Patter Recogn. Lett.* **27**(7), 773–780 (2006)
- Vemulapalli, R., Aravind, R.: Spatio-temporal nonparametric background modeling and subtraction. In: IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops, pp. 1145–1152 (2009)
- Kim, K., Chalidabhongse, T.H., Hardwood, D., Davis, L.: Real-time foreground-background segmentation using codebook model. *Real Time Imaging* **11**, 172–185 (2005)
- Guo, J.M., Hsia, C.H., Liu, Y.F., Shih, M.H., Chang, C.H., Wu, J.Y.: Fast background subtraction based on a multilayer codebook model for moving object detection. *IEEE Trans. Circuits Syst. Video Technol.* **24**(10), 1809–1821 (2013)
- Pal, A., Schaefer, G., Celebi, M.E.: Robust codebook-based video background subtraction. In: 2010 IEEE International Conference on Acoustics, Speech and Signal Processing, pp 1146–1149 (2012)
- Toyama, K., Krumm, J., Brumitt, B., Meyers, B.: Wallflower: principles and practice of background maintenance. In: Proceedings on the 7th IEEE International Conference on Computer Vision, Vol. 1, pp. 255–261 (1999)
- Sajid, H., Cheung, S.C.S.: Universal multimode background subtraction. *IEEE Trans. Image Process.* **26**(7), 3249–3260 (2017)
- Jiang, S.Q., Lu, X.B.: WeSamBE: a weight-sample-based method for background subtraction. *IEEE Trans. Circuits Syst. Video Technol.* (2017). <https://doi.org/10.1109/TCSVT.2017.2711659>
- St Charles, P.L., Bilodeau, G.A., Bergevin, R.: SuBSENSE: a universal change detection method with local adaptive sensitivity. *IEEE Trans. Image Process.* **24**(1), 359–372 (2015)
- Beaugendre, A., Goto, S.: Adaptive block-propagative background subtraction method for UHDTV foreground detection. *IEICE Trans. Fundam.* **98**(11), 2307–2314 (2015)

23. Beaugendre, A., Goto, S., Yoshimura, T.: Real-time UHD background modelling with mixed selection block updates. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **100**(2), 581–591 (2017)
24. Kim, H., Lee, H.J.: A low-power surveillance video coding system with early background subtraction and adaptive frame memory compression. *IEEE Trans. Consum. Electron.* **63**(4), 359–367 (2017)
25. Lim, L.A., Keles, H.Y.: Foreground segmentation using a triplet convolutional neural network for multiscale feature encoding. *arXiv preprint arXiv:1801.02225* (2018)
26. Zhao, Z.J., Zhang, X.B., Fang, Y.C.: Stacked multilayer self-organizing map for background modeling. *IEEE Trans. Image Process.* **24**(9), 2841 (2015)
27. Lim, L.A., Keles, H.Y.: Foreground segmentation using convolutional neural networks for multiscale feature encoding. *Pattern Recogn. Lett.* **112**, 256–262 (2018)
28. Zeng, W., Wang, K., Wang, F.Y.: A novel background subtraction algorithm based on parallel vision and Bayesian GANs. *Neurocomputing* (2020). <https://doi.org/10.1016/j.neucom.2019.04.088>
29. Changedetection.net (CDNET).: [Online]. <http://www.changedetection.net/>. Accessed 25 Jun 2018. (2018)
30. Wang, R., Bunyak, F., Seetharaman, G., Palaniappan, K.: Static and moving object detection using flux tensor with split Gaussian models. In: *Proceedings of IEEE Workshop on Change Detection* (2014)
31. Bianco, S., Ciocca, G., Schettini, R.: Combination of video change detection algorithms by genetic programming. *IEEE Trans. Evol. Comput.* **21**(6), 914–928 (2017)
32. Wang, Y., Jodoin, P.M., Porikli, F., Konrad, J., Benezeth, Y., Ishwar, P.: CDnet 2014: an expanded change detection benchmark dataset. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 393–400 (2014)
33. Goyette, N., Jodoin, P.M., Porikli, F., Konrad, J., Ishwar, P.: Changedetection.net: a new change detection benchmark dataset. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8 (2012)
34. Chang, H.J., Jeong, H.W., Choi, J.Y.: Active attention sampling for speed-up of background subtraction. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2088–2095 (2012)



**Woo-Young Jung** received his B.S., M.S., and Ph.D. degrees in Electronics and Communications Engineering department at Kwangwoon University. He is currently a CTO in Innovative Research and Analysis Inc. in Philippines and EPSOLUTE Co. Ltd. in Korea.



**Yong-Jin Jeong** received his B.S. degree in Control and Instrumentation Engineering from Seoul National University in 1983. He received his M.S. and Ph.D. degrees in Electrical and Computer Engineering from University of Massachusetts. He is currently a professor in the Dept. of Electronics and Communications Engineering, Kwangwoon University.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Vince Jebryl Montero** received his B.S. degree in Electronics Engineering from Ateneo de Davao University in 2014 and is currently studying for combined M.S. and Ph.D. degree in Electronics and Communications Engineering at Kwangwoon University.