Efficient Background Subtraction for Real-time Tracking in Embedded Camera Networks

Yiran Shen[†‡], Wen Hu[‡], Junbin Liu[‡⋆], Mingrui Yang[‡], Bo Wei[†‡] and Chun Tung Chou[†]

[†]University of New South Wales, Sydney, NSW, Australia
[‡]ICT Centre, CSIRO, Brisbane, QLD, Australia
[⋆]Queensland University of Technology, Brisbane, QLD, Australia

Abstract

Background subtraction is often the first step of many computer vision applications. For a background subtraction method to be useful in embedded camera networks, it must be both accurate and computationally efficient because of the resource constraints on embedded platforms. This makes many traditional background subtraction algorithms unsuitable for embedded platforms because they use complex statistical models to handle subtle illumination changes. These models make them accurate but the computational requirement of these complex models is often too high for embedded platforms. In this paper, we propose a new background subtraction method which is both accurate and computational efficient. The key idea is to use compressive sensing to reduce the dimensionality of the data while retaining most of the information. By using multiple datasets, we show that the accuracy of our proposed background subtraction method is comparable to that of the traditional background subtraction methods. Moreover, real implementation on an embedded camera platform shows that our proposed method is at least 5 times faster, and consumes significantly less energy and memory resources than the conventional approaches. Finally, we demonstrated the feasibility of the proposed method by the implementation and evaluation of an end-to-end real-time embedded camera network target tracking application.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Systems

General Terms

Algorithm, Experimentation, Performance

Keywords

Embedded Camera Networks, Background Subtraction, Compressive Sensing, Gaussian Mixture Models, Object Tracking

SenSys'12, November 6–9, 2012, Toronto, ON, Canada.

1 Introduction

Many recent real-time computer vision applications, such as object tracking, in embedded camera (or video sensor) networks require significant computation and energy resources. Robust background subtraction is typically the dominant factor. The aim of background subtraction is to detect whether the foreground is present in a newly acquired video frame. This is usually realised by using the knowledge of earlier video frames to learn a background model, and then applying statistical tests to decide whether the newly acquired frame is different from the background. A challenge for background subtraction is to differentiate between the foreground and subtle changes in the background, caused by events like illumination changes or moving tree branches. Moreover, a new challenge arises when background subtraction is to be used in embedded camera networks. The background subtraction algorithm must be computational efficient due to resource constraints of embedded platforms. In this paper, we present a new background subtraction method which is both accurate and computational efficient.

As mentioned earlier, one challenge for background subtraction is to differentiate the foreground from subtle changes in the background. This problem can be solved by modelling the background by some complex statistical models, such as, kernel density [11] or Gaussian density [29, 13, 26, 15] models. Among these, the mixture of Gaussians (MoG) [26] model is particularly popular because of its ability to deal with subtle illumination changes. The MoG method has been shown to be of good accuracy [2]. However, a complex statistical model can be a double-edged sword. On one hand, it gives good accuracy. On the other hand, it requires a lot of computational cost which makes real-time background subtraction on embedded platform challenging, if not infeasible. In this paper, we propose to resolve this tension by using the theory of compressive sensing.

Compressive sensing is a recently developed theory in signal processing. One of the key ideas behind compressive is that, if a signal x (or its coefficients in some domain) from a high dimensional space is sparse (or compressible), then it can be projected into a much lower dimensional signal y without losing much information. Therefore, instead of working in the high dimensional space, one can work on y in a much lower dimensional space, so as to achieve computational efficiency, as well as accuracy.

The contributions of this paper are four folds:

- We propose a novel background subtraction method that uses both compressive sensing (for dimensionality reduction) and MoG (for accuracy). We call it CS-MoG.
- We show, by using multiple real world datasets, that the accuracy of CS-MoG is comparable to MoG. We also show that CS-MoG is significantly more accurate than a number of other background subtraction methods.
- We implement both CS-MoG and MoG on an embedded platform, and show that CS-MoG is at least 5 times faster than MoG, and consumes significantly less energy and memory resources.
- We implement and evaluate an end-to-end multiple camera object tracking application based on CS-MoG, which demonstrates the feasibility of CS-MoG to operate in real-time scenario on embedded platforms.

The organisation of this paper is as follows. In Section 2, we provide technical background on compressive sensing and MoG. We then present our proposed method CS-MoG in Section 3. In Section 4, we evaluate the performance of CS-MoG, MoG and a number of other background subtraction methods. Section 5 presents results on running CS-MoG and MoG on an embedded camera platform. Section 6 presents related work and the conclusions are in Section 7.

2 Technical Background

In order to make this paper self-contained, this section provides technical background on compressive sensing and background subtraction using MoG.

2.1 Compressive Sensing

Compressive sensing [5, 9] is a recently developed method in signal processing. It proposes a method to recover a high dimension sparse signal from a small number of measurements. In order to explain compressive sensing, we consider a set of M simultaneous linear equations in N unknowns, written in matrix form:

$$y = \Phi x \tag{1}$$

where $x \in \mathbb{R}^N$, $y \in \mathbb{R}^M$ and $\Phi \in \mathbb{R}^{M \times N}$. The vector x is assumed to be unknown and the goal is to determine x. Both Φ and y are assumed to be known. The matrix Φ is a measurement matrix and y contains M measurements. Note that the measurements are linear combinations of the unknown vector x with the weights of the linear combinations specified in the matrix Φ. Compressive sensing is interested in the case where measurements are expensive to performed, so it is desirable to make M small compared with N. This means the system of linear equations is under-determined because $M < N$. We know from linear algebra that it is impossible to exactly recover (or solve for) x unless we impose additional conditions on x.

We now impose the requirement that the vector x is *sparse*. A vector x is said to be sparse if it contains very few non-zero elements. Let H denote the number of non-zero elements in x, then x is sparse if $H \ll N$. If $H > M$, then it is still impossible to recover x from y because there are more unknowns than equations. We therefore assume $H \leq M$ from now onwards.

Let us for the time being assume that we know the positions of the H non-zero elements in x but we do not know their values. In this case, we can re-write (1) as $y = \tilde{\Phi}\tilde{x}$ where $\tilde{x} \in \mathbb{R}^H$ contains the unknown non-zero elements of x and $\tilde{\Phi}$ contains the columns in Φ corresponding to the non-zero elements of x. It is now possible to exactly recover $\tilde{x}$ from y if $\tilde{\Phi}$ has rank H.

In general, the number of non-zero elements, as well as their positions and values, are unknown. It may still be possible to recover x but we have a combinatorial problem (in fact NP-hard) because the positions of the non-zero elements are not known. A striking result in compressive sensing is that it is still possible to recover x with high probability by solving the following ℓ_1 optimisation problem (which is solvable in polynomial time):

$$\hat{x} = \arg\min_{x \in \mathbb{R}^N} \|x\|_1 \quad \text{subject to } y = \Phi x \tag{2}$$

provided that the matrix Φ satisfies the Restricted Isometry Property (RIP) [4] and $M = O(H \log(N/H))$. Another striking result is that a number of classes of random matrices satisfy the RIP. For example, Φ satisfies the RIP if each element of Φ is ± 1 with equal probability, i.e. symmetric Bernoulli distribution.

In the terminology of compressive sensing, the matrix $\Phi \in \mathbb{R}^{M \times N}$, where $M < N$, is called a *projection matrix*. This name is used because the operation of transforming a higher dimensional vector into a lower dimensional vector is called a projection. In this case, the projection matrix Φ projects the higher (N) dimensional vector x onto the lower (M) dimensional vector y via the matrix multiplication $y = \Phi x$. We will refer to the elements in y as *projection values*.

Given that it is possible to recover x from y, an intuitive implication of the result in compressive sensing is that the projection values y contains almost all the information from the sparse vector x. It is also important to point out that proper choice of projection matrices plays a significant role in this result. In particular, matrices generated by symmetric Bernoulli distribution, which obey RIP, are able to "preserve" the information of x in y.

We have described the results on compressive sensing for sparse vectors x. The result can be extended to *compressible* vectors x in some transform domains, e.g. Discrete Cosine Transform (DCT) or wavelet domain [20]. We will not define compressibility precisely here but we instead state that it is a well known fact in image compression that blocks of pixels (typically 8×8 blocks of pixels expressed as a 64×1 vector) is compressible in DCT or wavelet domain.

In order to explain the intuition behind our work, we first recall the following two facts: (1) Blocks of pixels in an image are compressible; (2) Compressive sensing says that if a vector is compressible, then most of the information in the vector is also contained in its projection values. These two statements together imply that the information in a block of pixels is also contained in its projection values. Therefore, instead of using all the pixels in a block to decide whether it is foreground or not, we make the decision using the projection values of the pixels in the block because they contain almost the same information. This allows us to work with

data of significantly lower dimension, e.g., 8 projection values instead of 64 pixels. We will show that this dimensionality reduction results in a computational efficient background subtraction algorithm with little loss of accuracy.

2.2 Mixture of Gaussian Models for Background Subtraction

In [26], the authors proposed to use a MoG to model the background in background subtraction. In this method, the history of each pixel is modelled by a MoG consisting of K (typically chosen to be 3-5) Gaussian distributions. When a new video frame is presented, each pixel is compared with the MoG model for the corresponding pixel. If the new pixel value is within 2.5 standard deviation of any one of the K Gaussian distributions making up the MoG, then the pixel is considered a background candidate. A background candidate, afterwards, should be checked whether it belongs to a background distribution. The MoG for each pixel is updated for each frame. This update allows MoG to adaptively deal with noise and illumination changes which fixed threshold cannot handle.

The updating of the MoG model for a pixel is as follows. At time t, the MoG of each pixel consists of K Gaussian distributions. The k-th ($1 \leq i \leq K$) Gaussian is assigned a weight of $\omega_{k,t}$. If the new pixel value does not match any of the K Gaussians, the least probable distribution will be replaced by a new distribution with high variance and low weight; otherwise, the weight for the k-th Gaussian is updated as:

$$\omega_{k,t+1} = (1-\alpha)\omega_{k,t} + \alpha(G_{k,t+1}) \tag{3}$$

where α is the learning rate and $G_{k,t+1}$ is a binary variable whose value is 1 if the k-th Gaussian matches the new pixel and is zero otherwise. If the new pixel value x_{t+1} at time $t+1$ is accounted by, say the k-th, Gaussian distribution, its mean $\mu_{k,t}$ and variance $\sigma^2_{k,t}$ will be updated as:

$$\mu_{k,t+1} = \gamma x_{t+1} + (1-\gamma)\mu_{k,t} \tag{4}$$

$$\sigma^2_{k,t+1} = \gamma(x_{t+1} - \mu_{k,t+1})^2 + (1-\gamma)\sigma^2_{k,t} \tag{5}$$

where

$$\gamma = \frac{1}{\sqrt{2\pi}\sigma_{k,t+1}} \exp\left(-\frac{(x_{t+1}-\mu_{k,t+1})^2}{2\sigma^2_{k,t+1}}\right) \tag{6}$$

The probability that one of these K Gaussian distributions is the current background model is determined by the ratio of $\omega_{k,t}/\sigma_{k,t}$ at the current time t. When a new pixel value is available at time $(t+1)$, it will be checked if it belongs to any of the K distributions. If a new pixel does not match any of the K distributions, the least probable distribution will be replaced by a new distribution with mean equals to the new pixel value, and initial variance and weight. If the new pixel value matches any one of the K Gaussian distributions that models the pixel, the parameters of these distributions should be updated. After updating, the current K Gaussian distributions are sorted using the updated $\omega_{k,t+1}/\sigma_{k,t+1}$. According to this ratio and the prior information about the portion of the pixels accounted for the background (decided by the ratio of the background in camera's view), the number of background distributions in these K distributions is decided as,

$$N_b = \arg\min_n \left(\sum_{k=1}^{n} \omega_k > T_{h_b}\right) \tag{7}$$

where T_{h_b} is the portion of pixels accounted for by the background. This equation means that the first N_b distributions are chosen as the current background model. Therefore, the current pixel values that are located within 2.5σ of these N_b distributions will be marked as background. With the multimodal distributions, MoG can accommodate multiple background scenario well.

3 CS-MoG

The aim of this section is to describe CS-MoG, which is an accurate but yet computational efficient background subtraction method. The MoG background subtraction method (reviewed in Section 2.2) is able to deal with subtle changes in background because it models each pixel by a mixture of $3-5$ Gaussian distributions. However, this also makes MoG computational intensive to be used on embedded platforms. In order to simultaneously realise accuracy and computational efficiency, we propose CS-MoG. CS-MoG uses compressive sensing, which is what CS stands for, to reduce the dimensionality of the data while retaining much of the information. We then apply MoG to the reduced dimension data for background subtraction. This section is divided into two parts. We describe the steps of CS-MoG in Section 3.1 and justify the use of Gaussian mixture models for reduced dimension data in Section 3.2.

3.1 Steps of CS-MoG

Our method is divided into three steps. In the first step, the image is segmented into blocks of 8×8 pixels. (Note: We have experimented with different block sizes and they give similar results. We therefore assume a block size of 8×8 throughout this paper. The default block size in JPEG is also 8×8.) Projections based on compressive sensing are then computed for each block. In the second step, each projection value is modelled as a MoG to determine if the block contains some part of the foreground. We then fuse the results from all the projection values from a block to determine if it is a background or foreground block. The pixels in a background block are all background but the pixels in a foreground block can include both background and foreground. We call the second step foreground detection. The third step, which will be referred to as foreground refinement, is to identify which pixels in a foreground block is foreground. Thus, at the end of these three steps, each pixel in the image is labelled either as a foreground or background. Figure 1 shows the flow chart of the CS-MoG algorithm. Each dashed line box in the flow chart corresponds to a step of the algorithm. We will now describe each step in details.

3.1.1 Block Projections

Prior to performing background subtraction, we carry out a pre-processing step where we convert the video frames from RGB format to grey scale or intensity. This pre-processing step applies to all the background subtraction methods in this paper. The reason we do this is hardware limitation. We will see in Section 5 that the original MoG

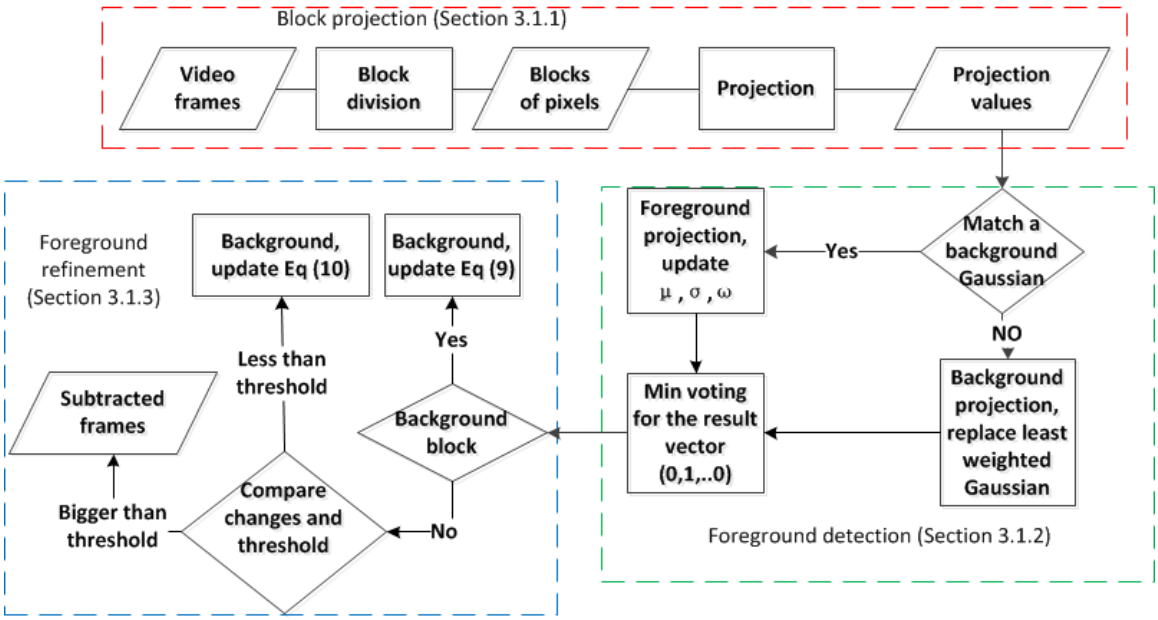


Figure 1. Flow chart of the Algorithm. Different components are grouped by dashed frames in different color

algorithm (described in Section 2.2) can only process on average about 3.6 grey-scale frames per second on the Blackfin BF-537 DSP camera while our proposed CS-MoG is able to process 17.7 frames per second. Note that it is straightforward to extend our proposed algorithm to RGB frames.

After pre-processing, we start the background subtraction process. The first step of CS-MoG is to divide the image into blocks of 8×8 pixels. After that, for each block, we form a 64×1 vector of the pixel values in a block and compute random projections of the vector in the same way as compressive sensing. The projection matrix we utilise is randomly generated at the beginning of a video sequence. Once it has been generated, the same projection matrix is used for each block for the entire video.

We consider two types of projection matrices, which we will call unbalanced matrices and balanced matrices. Each element of an unbalanced projection matrix is generated by a symmetric Bernoulli distribution of ± 1. A balanced projection matrix also consists of ± 1 at equal probability but in addition we require that each row must contain equal number of 1's and -1's. Therefore, the sum of the elements in each row of a balanced matrix is always zero. We will refer to CS-MoG that uses a balanced matrix (resp. unbalanced matrix) as CS-MoG-Balance (CS-MoG-Unbalance). In particular, we will show experimentally and analytically that CS-MoG-Balance gives better performance.

Since the operations to be carried out on each block of 8×8 pixels are identical, we will describe the operations on a block. We first stack the pixel values of the $n = 64$ pixels in a block into a $n \times 1$ vector that we call x. We assume that a $m \times n$ projection matrix Φ (which can be balanced or unbalanced) has also been generated. Note that m, which is the number of projection vectors, is a design parameter which we will study later on in section 4.2. Inspired by compressive sensing, we compute the projection:

$$y = \Phi x \tag{8}$$

The $m \times 1$ vector y contains the m projections values for this block. Given that the vector x (which contains the pixel values in a block) is compressible (Note: We know from image compression that the pixels in a block is compressible in DCT or wavelet basis [20].), we expect from compressive sensing that, for properly chosen value of m and projection matrix Φ, the m projection values in y contain almost all the information in x. This means that one can use the vector y, instead of x, to decide whether the block is foreground or not. Furthermore, we expect $m \ll n$ which means that we will be working with data of lower dimension. In fact, the results in section 4.2 show that $m = 8$ projections per block give good accuracy for background subtraction.

3.1.2 Foreground Detection

After computing the projections for each block, we need a method to determine whether this block contains some part of foreground according to projection values. To build a robust decision, we use MoG for the foreground detection step.

As we mentioned earlier, MoG can only process 3.6 frames per second on the Blackfin platform. In fact, the experiments in Section 5 shows that the Gaussian mixture computations take up almost all the processor resources. Therefore, computation efficiency can be improved if we can reduce the total number of Gaussian distributions that we use per frame.

Our main idea is to model *each projection value* by a mixture of K Gaussian distributions, which is similar to what MoG does for each pixel. (The choice of the parameter K will be discussed in Section 3.2.). Furthermore, we model each projection value independently and do not consider possible correlation between them. Following MoG, we consider a projection value to be a *background projection value*

candidate if it is within 2.5 standard deviations of one of the K Gaussian distributions that models the projection value; otherwise it is a *foreground projection value*. We apply this method to each of the m projection values in each block.

It is likely that the result of applying MoG to the m projection values will result in a mixture of background and foreground projection values. We therefore need a method to fuse the results. A number of fusion strategies are possible. Let us assume that the MoG test results in f foreground projection values out of all m projection values in block. We evaluated three fusion strategies: (1) Majority voting: the block is foreground if $f > \frac{m}{2}$; (2) Max voting: the block is foreground if $f = m$; and (3) Min voting: the block is foreground if $f \neq 0$, i.e. $f \geq 1$. Our evaluations (not shown here due to lack of space) show that min voting gives the best result. In the following, we will assume min voting is used.

3.1.3 Foreground Refinement

The foreground detection step so far works on the resolution of a block. This resolution may be sufficient for some applications but sometimes it is desirable to work with resolution at pixel level. We show how we can do that in this foreground refinement step.

We will assume that if a block is classified as the background, then all pixels in the block are background pixels. However, we cannot do the same for a foreground block. It is possible for a foreground block to contain both foreground and background pixels. This is especially true for those foreground blocks lying at the edge of the foreground. This means that we only need to work further on the foreground blocks. Since a video frame is expected to consist mainly of background blocks, the number of foreground blocks that we need to work with is likely to be small.

In order to determine which pixels in a foreground block is in fact foreground without introducing significant computation burden, we build a simple background learning strategy for each block. Our simulation and experiment results demonstrate that this simple method is accurate. Our pixel-scale background learning method can be described as follows. If a block X^b_{t+1} is marked as the background, then its pixel values are incorporated into the current background model B_t (at time t) of the block by using a learning rate α:

$$B_{t+1} = \alpha X^b_{t+1} + (1-\alpha)B_t \tag{9}$$

where B_{t+1} is the updated background model.

If the block X^f_{t+1} is marked as a foreground, then the background model B_t of this block will be updated with a background mask M_{t+1}

$$B_{t+1} = (\alpha X^f_{t+1} + (1-\alpha)B_t)_{M_{t+1}} \tag{10}$$

The background mask consists of indices of pixels which satisfy the following condition:

$$M_{t+1} = \text{Index}[(X^f_{t+1} - B_t) < \delta] \tag{11}$$

where δ is the threshold that accommodates a certain extent of noise and illumination change. This threshold is specified by the applications. For the datasets used in this paper and the experiments, a threshold around 10 (the pixel value is between 0 to 255) is suitable. A larger threshold should be applied if the illumination change is more severe. The function of the background mask is to prevent the foreground mistakenly being incorporated into the background model so that the background model for the foreground pixels at time $(t+1)$ will remain the same as that at time t. With these background model update processes, we are able to realise pixel-level background subtraction.

This completes the description of the three steps. We have evaluated this model on multiple datasets and found its accuracy to be good.

3.2 Parameter Choice for CS-MoG

The CS-MoG method that we have described comes with a number of different parameters. Two key parameters are the number of projections per block and the number of Gaussians K to model a projection value. We will study the number of projections later on in Section 4. In this section, we look at the choice of the parameter K which is the number of Gaussian distributions being used to model a projection value. In particular, we will show that a small value of $K = 3$ is needed. This is very encouraging because a small K means less computation burden.

Given that the same set of operations are applied to all blocks, it is sufficient to consider a block and the projection value obtained by one projection. Therefore, for this discussion, we consider a generic 8×8 block and we assume the value of i-th pixel x_i ($1 \leq i \leq 64$) is modelled by a random variable X_i. The projection value v is therefore a random variable $V = \sum_{i=1}^{64} \beta_i X_i$ where $\beta_i = \pm 1$ are the elements of a generic projection vector that we use in CS-MoG.

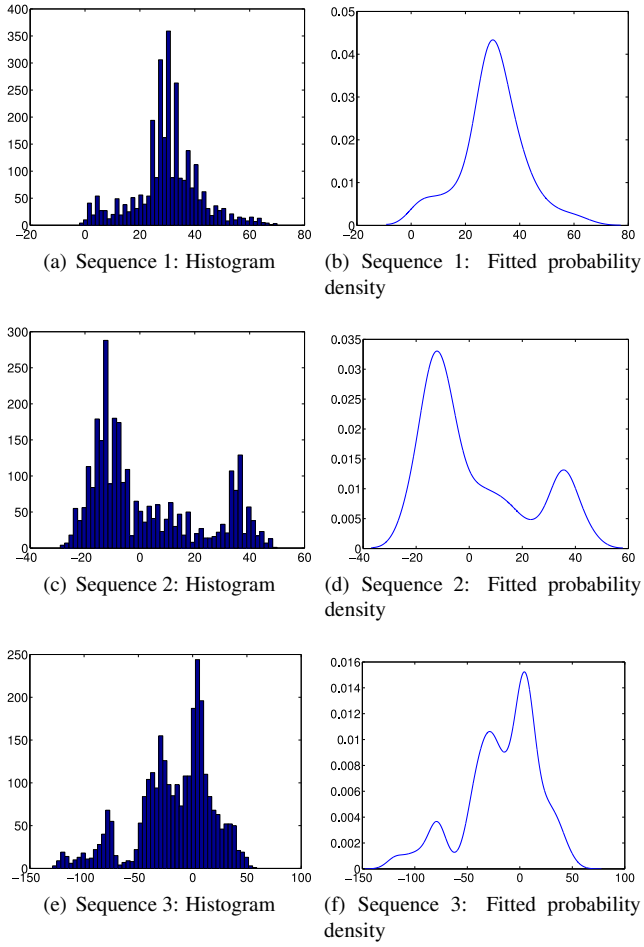
Many papers on background subtraction assume that each pixel is modelled by a Gaussian distribution. Let us for the time being assume that the random variable X_i is Gaussian distributed with mean μ_i and variance σ_i^2, i.e. $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. Since projection is a linear operation, standard result on Gaussian distribution says that the projection value v is also Gaussian distributed. In fact, $V \sim \mathcal{N}(\sum_{i=1}^{n} \beta\mu_i, \sum_{i=1}^{n} \sigma_i^2)$. Therefore, if we assume that the pixel values are Gaussian distributed, then we should choose $K = 1$ in CS-MoG.

However in [26], the authors state that it is not sufficient to model a pixel by a Gaussian distribution because such models can only deal with static background with illumination change. In order to be able to accommodate complex background with multiple surfaces and illumination changes, we need to model each pixel by a mixture of Gaussians because these illumination changes cannot always be captured by the same Gaussian. This is precisely the motivation for MoG background subtraction method in [26]. This also implies that a better pixel model is given a mixture of Gaussians, rather than a single Gaussian.

Let us consider the case that each random variable X_i is a mixture of Q Gaussians. Since the probability distribution of a sum of random variables is equal to the convolution of the probability distributions of the random variables, it can be shown that the random variable V is still a mixture of Gaussians. However, the number of Gaussians needed can be as large as Q^{64}. The use of such a large number of Gaussians is certainly not practical. We therefore investigate whether

it is possible to use a small number of Gaussians to model a projection value.

For our investigation, we use three datasets, or video sequences. (The details of these datasets are described in Section 4.1.) The same method of investigation is applied to all three datasets. Consider a video sequence consisting of F frames, indexed by $f = 1, ..., F$. Each frame consists of B 8×8-block, indexed by $b = 1, .., B$; it is assumed that the b-th block is always located in the same location within a block. We then generate P projection vectors, which are indexed by $p = 1, ..., P$. By computing the projection values of the p-th projection vector with the b-th block in frames $f = 1, ..., F$, we obtain a sequence of F projection values. We do this for each combination of p and b, giving altogether BP sequences of projection values per dataset. Figure 2 shows the histograms and fitted probability densities of three different sequences of projection values. Visual inspection suggests that these three sequences can be modelled by a probability distribution with one to three Gaussians.



(a) Sequence 1: Histogram (b) Sequence 1: Fitted probability density

(c) Sequence 2: Histogram (d) Sequence 2: Fitted probability density

(e) Sequence 3: Histogram (f) Sequence 3: Fitted probability density

Figure 2. This figure shows distributions of three sequences of projection values. Figures on each row correspond to one sequence with 3000 projection values. The x-axis is the projection value. The figure on the left shows the histogram, i.e. y-axis is frequency of each bin. Figures on the right show the fitted probability density.

In order to systematically determine the number of Gaussian distributions needed to approximate a sequence of projection values, we use the kernel density estimation method in [3] to calculate the number of Gaussians required. We do this for all three video sequences and the results are shown in the table 1. The results show that, for each dataset, over 99.9% of the projection value sequences can be modelled by no more than 3 Gaussians. We therefore choose the value of K to be 3 in our CS-MoG method. Note that this is a very encouraging result, especially if we want to implement our CS-MoG method on embedded platforms. Consider the original MoG where each pixel is modelled by 3 Gaussians, which means we need 64×3 Gaussians per block. For our CS-MoG, we show later that 8 projections per block is sufficient; since each projection value needs 3 Gaussians, the number of Gaussians needed per block is 8×3, which is a reduction by a factor of 8.

	One	Two	Three	More
Datasets 1	99.94 %	0.06 %	0%	0%
Datasets 2	89.58%	10.06%	0.35%	0.01%
Datasets 3	76.28%	19.12%	3.65%	0.095%

Table 1. Number of Gaussians required for approximating distribution of projection.

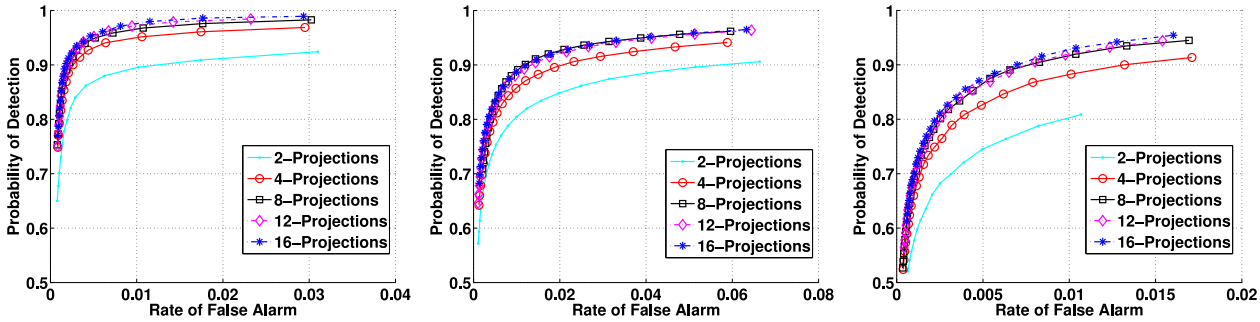
4 Performance Evaluation

4.1 Goals, Metrics and Methodology

The goals of our evaluation is to demonstrate whether CS-MoG 1) achieves the best subtraction accuracy among MoG-based efficient background subtraction algorithms, 2) improves the performance of background subtraction against another compressive sensing inspired algorithm CSBS [6], and 3) obtains a better capability to deal with illumination change especially with balanced matrix.

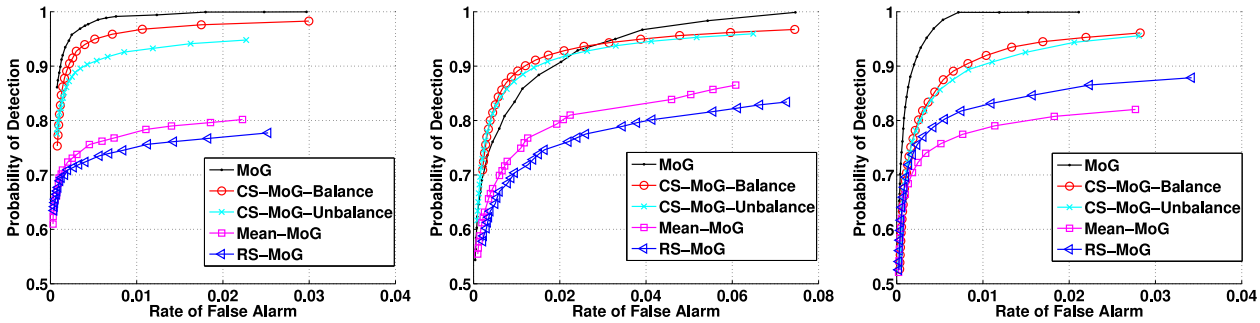
We use three different datasets to evaluate the performance of various background subtraction algorithms. The first dataset (dataset 1) is a private dataset from our laboratory for monitoring a footpath. Datasets 2 and 3 are, respectively, VS-PETS'2003 and PETS'2001, from `http://www.cvg.rdg.ac.uk/`. Dataset 2 is on a football match while dataset 3 is from monitoring people and vehicles outdoor.

Given that the spirit of CS-MoG is to use projections to reduce the dimension of input to MoG, we consider two other methods to realise reduction in dimension but not using projections. The first method is called Random Sampling MoG (RS-MoG). RS-MoG is identical to CS-MoG except that RS-MoG does not compute m projections. Instead, RS-MoG uses m random pixels in a block to decide whether that block is foreground or not. The second method is called Mean-MoG. For Mean-MoG, we divide each block into m sub-blocks and compute the mean pixel values of each sub-blocks. These mean values are input to MoG for foreground detection. Note that CS-MoG, RS-MoG and Mean-MoG use, respectively, m projections, m pixel values and m mean values per block for foreground detection. The dimensionality reduction for these methods are therefore identical.



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

Figure 3. ROC curves for CS-MoG-Balance for different number of projections for the three datasets.



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

Figure 4. ROC curves for different MoG-based methods for the three datasets.

In this paper, we regard a pixel in the foreground (resp. background) as a positive (negative) event. A false positive means a genuine background pixel is incorrectly detected as a foreground. We express the performance of various methods by using the receiver operating characteristic (ROC) curve. The vertical axis of the ROC curve is the probability of detection (P_D) which is the total number of true positives divided by the number of foreground pixels (positive events) in ground truth. The horizontal axis of the ROC curve is the rate of false alarm (F_A) which is the number of false alarms (or false positives) divided by the number of background pixels (or negative events) in ground truth.

4.2 Impact of Number of Projections on the ROC for CS-MoG Balance

We first evaluate the impact of the number of projections on the ROC for CS-MoG-Balance. We apply CS-MoG-Balance to 400 consecutive video frames in each dataset. For each dataset, we use the following number of projections: 2, 4, 8, 12 and 16. The results for the three datasets are shown in Figure 3. We can make two observations from these figures. Firstly, the performance of CS-MoG-Balance improves with an increasing number of projections. Secondly, the performance improvement diminishes when 8 or more projections are used. These observations can be explained by the fact the amount of information increases with the number of projections. However, most of the information in a block can be captured by 8 projections. Given these observations, we will use $m = 8$ projections for CS-MoG for the rest of this performance evaluation.

4.3 Performance of CS-MoG, MoG and other MoG based algorithms

In this section, we compare the performance of five background subtraction methods. They include pixel based MoG [26], CS-MoG-Balance and CS-MoG-Unbalance, which are described in Section 2.2, as well as RS-MoG and Mean-MoG introduced in Section 4.1.

We apply these five methods to 400 consecutive video frames from each dataset. The value of m is chosen to be 8 for CS-MoG-Balance, CS-MoG-Unbalance, RS-MoG and Mean-MoG. The RoC curves for these five methods are plotted in Figure 4 for the three datasets.

We can see from these figures that out of all the methods that use dimensionality reduction, CS-MoG-Balance gives the best performance. It may not be surprising that the simple methods such as RS-MoG and Mean-MoG do not perform that well. The observation that CS-MoG-Balance performs better than CS-MoG-Unbalance deserves further investigation. This is the topic of Section 4.5.

We see from these figures that the performance of MoG and CS-MoG-Balance are comparable except for the third dataset. It is probably not surprising that MoG has a better performance most of the time because it maintains complete information on each pixel. However, the better performance of MoG comes at the expense of a high computation cost. We will show in Section 5, by implementing both MoG and CS-MoG on an embedded platform, the computation time for

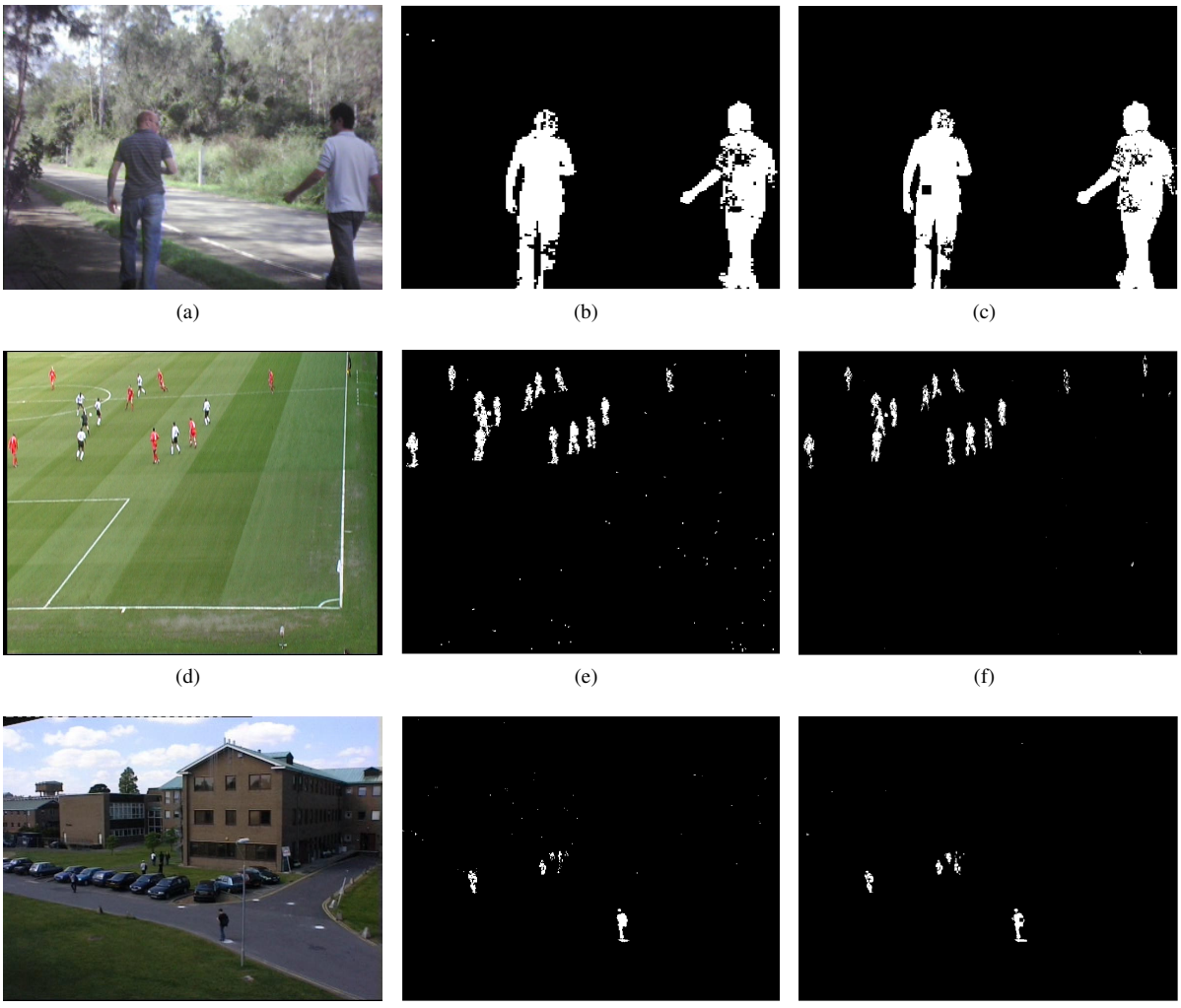


Figure 5. Sample background subtraction outputs for MoG (middle column) and CS-MoG-Balanced (rightmost column). The i-th ($i = 1,2,3$) row contains the results from the i-th dataset. The original frames are in the leftmost column.

MoG is 5 times slower than that of CS-MoG and real-time background subtraction with MoG is not feasible. Therefore, when we take into account both performance and resource constraints on embedded platforms, CS-MoG-Balance is a better choice compared with MoG.

We will conclude this comparison of various MoG-based background subtraction methods by showing a number of sample frame outputs. We first compare MoG and CS-MoG-Balance. Figure 5 shows the background subtraction results from these two methods for one frame from each dataset. The middle column of the figure shows the result from MoG while those for CS-MoG-Balance are shown in the rightmost column. It can be seen that the results are comparable.

Figure 6 shows the background subtraction result from all the five MoG-based methods for a particular frame from the first dataset. In addition, we have also shown in the caption of each sub-figure the number of miss detections F_n and the number false positives (false alarms) F_p. It demonstrates that CS-MoG, especially with balance matrix, achieves a significantly better performance for dealing with the false alarm than original MoG (94 vs 286) meanwhile preserving a comparably accurate foreground shape (94 vs 88). Moreover, Mean-MoG and RS-MoG have capability in dealing with false alarm as well (127 and 141). However, their performance for the foreground detection is evidently deteriorated (147 and 175), such as the shoulder of the person on the right in Figure 6(e) and the legs of the person on the right in Figure 6(f). To sum up, CS-MoG achieves the closest foreground detection accuracy compared to the original MoG, meanwhile, it dramatically eliminates the amount of false alarms.

4.4 Comparing CS-MoG-Balance and CSBS

In this section, we will compare the performance of CS-MoG-Balance against another compressive sensing based background subtraction method, namely Compressive Sensing for Background Subtraction (CSBS) [6]. There are many

(a) Original image (b) F_n: 94, F_p: 94 (c) F_n: 93, F_p: 166

(d) F_n: 88, F_p: 286 (e) F_n: 147, F_p: 127 (f) F_n: 175, F_p: 141

Figure 6. Sample background subtraction output for various MoG based methods. (a) Original frame, (b) CS-MoG-Balance, (c) CS-MoG-Unbalance, (d) Original MoG, (e) Mean-MoG, (f) RS-MoG.

differences between CS-MoG-Balance and CSBS but the two main differences are: (1) CSBS assumes that each projection value can be modelled by a Gaussian distribution, however CS-MoG-Balance assumes a Gaussian mixture; (2) CSBS obtains the foreground by solving an ℓ_1 optimisation problem but CS-MoG-Balance does not need to solve any optimisation problem. Because our method is based on a more sophisticated background model and will not include ℓ_1 optimisation, it is expected to be more robust and efficient compared with CSBS.

Figure 7 shows the RoC curves for both CSBS and CS-MoG-Balance for dataset 1. It can readily be seen that CS-MoG-Balance performs significantly better than CSBS. This is due to the fact that CSBS assumes that each pixel is Gaussian distributed and therefore cannot deal with subtle illumination and repetitive background changes. The number of projections used for both CS-MoG-Balance and CSBS is 8 per 8×8 block.

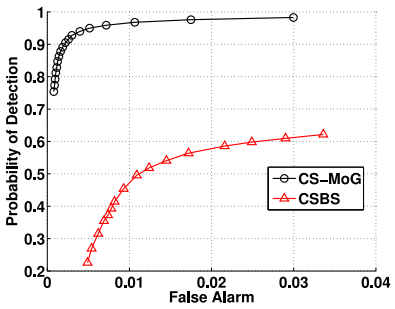


Figure 7. the ROC curves for CSBS and CS-MoG.

4.5 Balanced versus Unbalanced Projection Matrices

The experiments in Section 4.3 show that CS-MoG-Balance has a better performance compared with CS-MoG-Unbalanced. Closer investigation shows that the performance difference between these two methods is due to the way they handle illumination change. We claim that balance projection matrices have a better capability to address illumination change effects than unbalanced projection matrices. The reason simply comes from its "balance" feature. In order to explain this, we investigate the effect of projection matrices using two different illumination change models.

The first and simpler illumination change model is taken from [22]. The ith pixel value measured by the camera depends on the illumination reaching the surface (l_i) and its albedo feature a_i. Under the Lambertian assumption, the pixel value is: $x_i = l_i a_i$. The albedo feature a_i is determined by the feature of the surface. Thus, a change in the source of illumination will affect the pixel value x_i through l_i. If the source of illumination is large and far away, the illumination reaching a small surface area can be assumed to be constant over the limited size of the surface. Given that our proposed CS-MoG considers a small block of pixels at a time, we can therefore assume that l_i is constant over a block. Consequently, the illumination change measured by the camera within a block is only determined by the albedo feature a_i.

Let us consider the case that the block experiences a constant shift in illumination level Δ, i.e., $l_i = \Delta$ for all pixels in a block. Similar to Section 3.2, we use β_i (which equals to ± 1) to denote the elements of the projection vector. The

change in projection value δv is given by: $\delta v = (\sum_i \beta_i a_i)\Delta$.

Consider the case that the surface feature is the same within the block, then a_i takes the same value for all pixels in a block. If the matrix is balanced, the change in projection value δv is zero because $\sum_i \beta_i$ is always zero for a balanced projection matrix. Therefore a constant change in illumination will not change the projection value. However, if the surface feature is not even, which happens when the block is at the edge of the foreground, the illumination change will not be cancelled out. Although the illumination change is not always exactly constant on the whole block, especially near the edge of the foreground, a balanced projection matrix still has a better ability to deal with the illumination change compared with a unbalanced one.

More precise results can be obtained by assuming a statistical model for illumination change. By assuming that the illumination change is Gaussian distributed, we show analytically in Appendix A that a balanced projection matrix maximises the probability of correct detection of background.

5 Experiments on an Embedded Platform

Two sets of experiments were conducted to evaluate the performance of the proposed CS-MoG. The first set aimed at benchmarking the performance of the proposed algorithm against the original MoG in terms of computational time, memory usage and energy consumption under different configurations. We did not include CSBS in the comparison because CSBS requires solving $\ell 1$ optimisation problem to obtain the results in pixel domain which is highly computation intensive. Our platform cannot handle it real-time. The second set of experiments demonstrate the feasibility of CS-MoG on embedded platforms with a end-to-end distributed multi-camera object tracking application.

5.1 Comparison with Original MoG

We implemented both the original MoG and CS-MoG on a Blackfin BF-537 DSP based wireless camera node (see Figure 8(a)), which is similar to the platform used in [7] but attached to a mote with a Zigbee radio (Atmel AT86RF212 transceiver) and an Atmel Atmega1281 micro-controller. The operating system in the Blackfin DSP is Analog Devices' VDK kernel and the development environment is the VisualDSP++. This combination has been shown to have better performance compared to open source options such as μCLinux according to [7]. The operating system in the Zigbee radio mote, which is attached to the Blackfin DSP, is TinyOS 2.x. We also ported Berkeley version of 6LoWPAN (BLIP) as the communication stack for the wireless camera node.

Table 2 shows the performance (computation time, memory requirement and energy consumption) results of both the original MoG and CS-MoG at different image resolutions. The run-time results are computed as the mean of 100 consecutive frames. Table 2 does not contain speed and memory results for MoG under 640×480 resolution due to insufficient memory to run MoG for this resolution. In terms of processing speed, it can be seen from Table 2 that CS-MoG is approximately five times faster than the original MoG. Furthermore, CS-MoG requires approximately a quarter of memory compared to that of the original MoG. Therefore, CS-MoG consumes much less resources compared to the original MoG, representing a significant improvement that is crucial to embedded computer vision applications because other algorithms, e.g., face detection and recognition for identifying each tracked target, can now run on the remaining processing power. Table 2 also shows that the MoG component dominates the computational time for CS-MoG and in fact it consumes as high as approximately 85% of total computation time in CS-MoG. This observation justifies the proposed dimension reduction using compressive sensing, because, compared to the computation time reduction in the MoG component, the overhead introduced by computing the projections and foreground refinement is negligible.

Table 2. Resource consumption of different methods on different image resolution. CS refers to computing the projections

	MoG 320×240	CS-MoG 320×240	CS-MoG 640×480
Initialization (ms)	0.88	0.88	3.7
CS (ms)	–	3.06	12
MoG (ms)	280	49	210
Voting (ms)	–	0.25	1.02
Refinement (ms)	–	3.5	21.3
Total (ms)	280.88	56.6	278.28
Energy (mJ)	315.15	63.50	312.23
Memory (byte)	1,382,448	366,146	1,464,146

5.2 CS-MoG for Real-Time Distributed Object Tracking

To demonstrate the feasibility of CS-MoG for embedded computer vision applications, we further implemented an end-to-end distributed multi-camera tracking application on the BF-537 camera platform introduced in Section 5.1.

In the experiments, three wireless cameras were set-up in an approx. $4m \times 4m$ area with overlapping coverage of the ground. The cameras communicate with a server using the BLIP and IPv6 network. We use a toy train (see Fig. 8(b)) as the target in the experiments in order to collect high-precision (in cm) ground truth information. In this setting, the size of foreground taken up by the train is smaller than that in dataset 1 (see Fig. 5(a)) but is larger than that in datasets 2 and 3 (see Fig. 5(d) and 5(g)). The experimental set-up is shown in Fig. 9.

We further deployed a number of tags on the ground which are needed to compute the ground plane homographies [28] of each camera. A homography is a projective transformation that maps the coordinates from one plane to another, which in this case are the camera's image plane and the ground. With the computed homographies, we were able to obtain the calculated locations of the target and the ground truth in the ground coordinates with high-precision.

The target (train) moves along a track within the area of interest. All cameras continuously process incoming frames using CS-MoG to firstly segment out the moving foreground, which is then passed into a connected component analysis

that outputs the centroid of the moving object. The centroids are taken as the object's locations in the image coordinates. To conserve resources (in terms of radio bandwidth and energy consumption), packets that contain these object's locations (in the image coordinates) are only transmitted to the server from a camera when the object is in the camera's field of view. When the server obtains a location message along with a camera ID (e.g., IPv6 address), it will calculate the locations in the ground coordinates using the corresponding homography of the camera.

We ran the tracking experiment on two tracks of different shapes and the target (train) made three laps on each track (see Fig. 10(a) and 10(e)). The right columns of Fig. 10 shows the tracking results of each lap for the two tracks. The black crosses in Fig. 10 are the ground truth in the ground coordinates and the results from three camera nodes are shown as crosses of different colours (red, green, blue). Furthermore, we calculated the mean and standard deviation of the distances from the estimated target locations to the ground truth and the results are shown in the caption of the figures. Overall, we achieved high precision with less than 1% (less than $4cm$) target localization errors relative to the size of the area of interest ($4m \times 4m$).

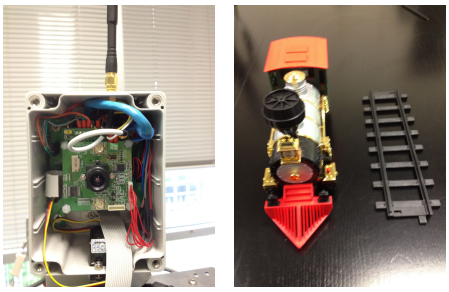
If the original MoG were used in our experiment, we would be able to only achieve a frame rate of approx. 3 frames per second. A lower frame rate would result in a higher tracking error in estimating the trajectory of the train. The average tracking error for MoG, for the first and second tracks, are, respectively, 4.9cm and 4.7cm. These results are obtained from comparing the tracking output of MoG against that of the ground truth given by homography. This represents 40% and 42% decrease in accuracy compared to CS-MoG results respectively. The high frame rate of CS-MoG can be matched by applying a downsampling strategy to the original MoG, where smaller images are processed. However, as shown by the Mean-MoG curves in Figure 4, this strategy will penalize the accuracy of the foreground segmented by MoG and thus the tracking performance.

Similar to the result reported in [1], BLIP achieved more than 99% packet reception rates in our experiment. The size of the communication payload was 10 bytes, and the average network goodput was 727 bps in experiment 1 and 1,066 bps in experiment 2 respectively. The peak network goodput was 1,841 bps in experiment 2 when three cameras nodes had overlapping view of the target. The end-to-end communication latency was 45.22 ± 0.74ms, which was sufficient for real-time tracking applications. For the larger networks, previous work reported approximately 120, 40 and 35 kbps for one-hop, two-hop and three-hop links respectively [1]. Therefore, we believe that BLIP can support up to 60 one-hop nodes or 20 multiple-hop nodes in our experiment setting.

6 Related Work

In this section, we cover four topics related work: background subtraction methods, compressive sensing in computer vision, compressive sensing in sensor networks and embedded camera networks.

MoG [26] has been one of the most popular background



(a) Blackfin DSP camera (b) Train with track

Figure 8. Camera and tracking target in experiment.

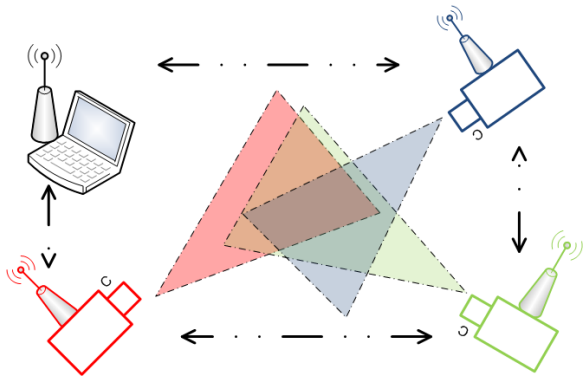
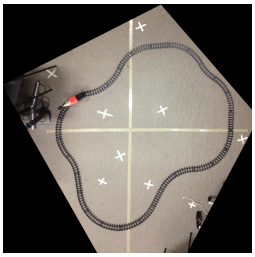
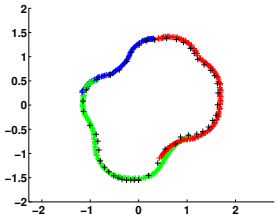


Figure 9. Target tracking experiment setting.

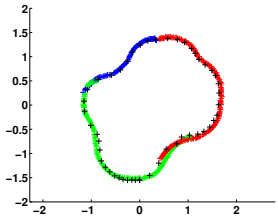
subtraction techniques in computer vision because of its robustness to subtle illumination changes. Its bottleneck is its computational intensity because of the need to compute and update the Gaussian mixtures. Instead of using a fixed number of Gaussian mixtures as in [26], the work in [33] adaptively determines the number of Gaussians for each pixel. This results in a more computational efficient procedure. The comparison in [33] shows that the adaptive procedure is $2\%-30\%$ faster compared with original MoG. However, our experiments in Section 5 shows that our proposed CS-MoG is 5 times faster. Another example of improving the efficiency of MoG is in [27]. In this work, it simplifies the learning update of the Gaussian mixtures and instead of using $\frac{\omega}{\sigma}$ to order the Gaussians, it simply uses ω instead. These simplifications can decrease the computation time of MoG by 1.6 times. However, these simplifications may not be suitable for some situations because the reorder condition is only based on ω, which may increase or decrease so quickly that a slowly moving target may be mistakenly incorporated into background or wrongly removed from the current background model. Our proposed CS-MoG performs foreground detection in two stages. The first of which is block-based foreground detection step (Section 3.1.2) and subsequently pixel-based foreground refinement step (Section 3.1.3). A classical method to perform background subtraction efficiently is to use block-based methods. An example is in [25]. This block-based background subtraction method divides a frame into 8×8 blocks and then computes a feature vector with 8 elements. For foreground detection,



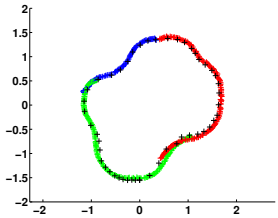
(a) Top view of the track shape 1



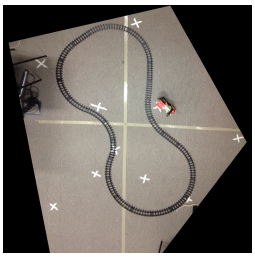
(b) 0.036m±0.018m



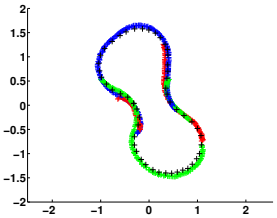
(c) 0.034m±0.018m



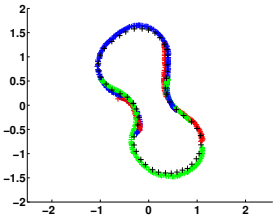
(d) 0.035m±0.017m



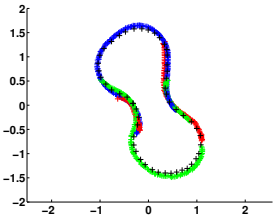
(e) Top view of the track shape 2



(f) 0.032m±0.020m



(g) 0.034m±0.021m



(h) 0.033m±0.021m

Figure 10. Target tracking experiment set-up and results. The black cross are the ground truth and the other different colour cross represent the results from different camera nodes.

it uses a block-scale background training set for comparison using normalised vector distance as the criterion. The method can be used as an assistant to traditional pixel based background model like MoG to increase the accuracy. However, its accuracy is related to the size of training set. Also, the algorithm has to traverse the whole dataset to find the closest fit. When the dataset is large, the algorithm will be computational intensive. However, our CS-MoG does not require any training set.

Compressive sensing has been an active area of research recently. There is much work in the area and we will limit this review to work in compressive sensing that is related to computer vision and sensor networks. One of the most important breakthroughs in the computer vision with compressive sensing is the invention of compressive sensing based imaging hardwares. Two examples are single pixel camera [10] and random convolution camera [24, 16]. These cameras exploit the theory of compressive sensing. Instead of sampling pixel-wise, they use projections as the measurements. As a result, the sampling requirements of these cameras are significantly lower. Other research work in using compressive sensing in computer vision include the CSBS background subtraction algorithm [6] ,object tracking and 3-D reconstruction [23, 8] .

Sensor networks are resource constraint while compressive sensing is capable of significantly reducing the sampling rate and data dimension. It is a hot-spot in researching on applying compressive sensing and its relevance: sparse representation in sensor networks. Compressive sensing can be used as an efficient sampling strategy. In [31], the author considered the problem of monitoring soil moisture with a wireless sensor networks. With compressive sensing, they achieve high accuracy at no more than 10% of the traditional sampling rate. Also the sparse representation is attracting increasingly attentions with compressive sensing. One of the examples is [21]. In this paper, the authors deal with cross-correlation problem (which is widely used in sensor networks) efficiently via sparse representation

The research area of embedded camera networks presents a lot of research challenges because the capturing, processing and communicating of images and videos are energy intensive operations. Therefore, a research focus of embedded camera networks is to optimise the energy consumption of their various operations. Communication is one of the most energy demanding operations in embedded camera networks [18]. In [18], the authors compare the energy consumption between two image transmission methods: (1) Direct transmission of compressed image; and (2) Compression of images and its subsequent transmission. The comparison is performed on various types of embedded platforms. The result varies according to the processor-radio combination of the embedded platform. Their results therefore provide a reference method to choose between these two transmission methods for different embedded platforms. Due to significant energy and bandwidth consumption of transmitting images, recent work in embedded sensor networks avoids transmitting large amount of image data. Instead, these papers perform the image or video processing on the embedded platform and communicate only the results, which generally have smaller data size and thus a lower bandwidth requirement. An example of such work is [14] where distributed camera localisation is used to determine the location and orientation of each camera. The distributed processing reduces inter-device communication and also the communi-

cation between device and base station. Another example is distributed image searching [32]. This work divides and shifts the function of the central search engine to end nodes in which images from different sensors can be captured, stored, searched and queried locally. Distributed processing is also used in [19] to exploit the overlap of view points of adjacent cameras to infer the location of the cameras. The paper [30] uses a camera network to estimate human posture. The key points (e.g., joints) are computed locally while the posture is estimated at the base station. There is also recent work in using camera networks for object tracking [17, 12]. However their focus is different. In [17], the tracking system makes a localised decision to switch between simple or elaborate background subtraction methods according to the application requirements to decrease resource consumption of camera networks. However, [17] does not propose any new background subtraction methods whereas our work proposes a new background subtraction method which is both accurate and computational efficient. Lastly, the work in [12] deals with the occlusion problems in object tracking, and applies local processing and clustering to conserve communication bandwidth and energy; its focus is therefore different from ours.

7 Conclusions

In this paper, we address the challenge of performing background subtraction, both accurately and efficiently, on embedded camera networks. Traditional background subtraction algorithms, though accurate, are not computational efficient because complex statistical models are needed to capture subtle illumination changes. To address this computational bottleneck, we use compressive sensing to reduce the dimensionality of the data while retaining the information content. This results in a computational efficient and yet accurate background subtraction algorithm. Our experiments show that the accuracy of our proposed algorithm is comparable to that of traditional algorithms but is five times more efficient. Furthermore, we show that our proposed background subtraction algorithm can accurately track a moving object in real-time in an embedded camera network.

8 References

[1] M. Afanasyev, D. O'Rourke, B. Kusy, and W. Hu. Heterogeneous traffic performance comparison for 6lowpan enabled low-power transceivers. In *Proceedings of the 6th Workshop on Hot Topics in Embedded Networked Sensors (HotEmNets '10)*, pages 10:1–10:5, New York, NY, USA, 2010. ACM.

[2] Y. Benezeth, P. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Review and evaluation of commonly-implemented background subtraction algorithms. In *19th International Conference on Pattern Recognition (ICPR '08)*, pages 1–4, Dec 2008.

[3] Z. I. Botev, J. F. Grotowski, and D. P. Kroese. Kernel density estimation via diffusion. *Annals of Statistics*, 38:2916–2957, 2010.

[4] E. Candes and E. J. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique.*, 346:589–592, 2008.

[5] E. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52:489–509, Feb 2006.

[6] V. Cevher, A. Sankaranarayanan, M. F. Duarte, D. Reddy, R. G. Baraniuk, and R. Chellappa. Compressive sensing for background subtraction. In *Proceedings of the 10th European Conference on Computer Vision (ECCV '08)*, pages 155–168, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] P. Corke, J. Liu, D. Moore, and T. Wark. Design and evaluation of an image analysis platform for low-power, low-bandwidth camera networks. In *Workshop on Applications, Systems, and Algorithms for Image Sensing (ImageSense '08)*, Nov 2008.

[8] M. Cossalter, M. Tagliasacchi, and G. Valenzise. Privacy-enabled object tracking in video sequences using compressive sensing. In *6th IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS '09)*, pages 436–441, Sept 2009.

[9] D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52:1289–1306, April 2006.

[10] M. Duarte, M. Davenport, D. Takhar, J. Laska, T. Sun, K. Kelly, and R. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25:83–91, March 2008.

[11] A. M. Elgammal, D. Harwood, and L. S. Davis. Non-parametric model for background subtraction. In *Proceedings of the 6th European Conference on Computer Vision (ECCV '00)*, pages 751–767, London, UK, 2000. Springer-Verlag.

[12] A. O. Ercan, A. El Gamal, and L. J. Guibas. Object tracking in the presence of occlusions via a camera network. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, pages 509–518, New York, NY, USA, 2007. ACM.

[13] N. Friedman and S. Russell. Image segmentation in video sequences: a probabilistic approach. In *Proceedings of the 13th conference on Uncertainty in artificial intelligence (UAI '97)*, pages 175–181, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[14] S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankar. Distributed localization of networked cameras. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN '06)*, pages 34–42, New York, NY, USA, 2006. ACM.

[15] E. Hayman and J.-O. Eklundh. Statistical background subtraction for a mobile observer. In *9th IEEE International Conference on Computer Vision (ICCV '03)*, pages 67–74, Oct 2003.

[16] L. Jacques, P. Vandergheynst, A. Bibet, V. Majidzadeh, A. Schmid, and Y. Leblebici. CMOS compressed imaging by random convolution. In *34th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '09)*, pages 1113–1116, April 2009.

[17] A. Kamthe, L. Jiang, M. Dudys, and A. Cerpa. Scopes: Smart cameras object position estimation system. In *Proceedings of the 6th European Conference on Wireless Sensor Networks (EWSN '09)*, pages 279–295, Berlin, Heidelberg, 2009. Springer-Verlag.

[18] D.-U. Lee, H. Kim, S. Tu, M. Rahimi, D. Estrin, and J. D. Villasenor. Energy-optimized image communication on resource-constrained sensor platforms. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, pages 216–225, New York, NY, USA, 2007. ACM.

[19] F. Li, J. Barabas, and A. L. Santos. Live photo mosaic with a group of wireless image sensors. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (Sensys '09)*, pages 359–360, New York, NY, USA, 2009. ACM.

[20] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41:3397–3415, Dec 1993.

[21] P. Misra, W. Hu, M. Yang, and S. Jha. Efficient cross-correlation via sparse representation in sensor networks. In *Proceedings of the 11th International Conference on Information Proceeding in Sensor Networks (IPSN '12)*. ACM, 2012.

[22] J. Pilet, C. Strecha, and P. Fua. Making background subtraction robust to sudden illumination changes. In *European Conference on Computer Vision (ECCV '08)*, pages 567–580. Springer Berlin Heidelberg, 2008.

[23] D. Reddy, A. Sankaranarayanan, V. Cevher, and R. Chellappa. Compressed sensing for multi-view tracking and 3-d voxel reconstruction. In *15th IEEE International Conference on Image Processing (ICIP '08)*, pages 221–224, Oct 2008.

[24] J. Romberg. Compressive sensing by random convolution. *SIAM Journal on Imaging Sciences*, 2:1098–1128, 2009.

[25] D. Russell and S. Gong. A highly efficient block-based dynamic background model. In *3th IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS '05)*, pages 417–422, Sept 2005.

[26] C. Stauffer and W. Grimson. Adaptive background mixture models

for real-time tracking. In *12th IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '99)*, pages 246–252, 1999.

[27] Z. Tang and Z. Miao. Fast background subtraction and shadow elimination using improved gaussian mixture model. In *IEEE International Workshop on Haptic, Audio and Visual Environments and Games (HAVE '07)*, pages 38–41, Oct 2007.

[28] P. H. S. Torr and A. Zisserman. Feature based methods for structure and motion estimation. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice (ICCV '09)*, pages 278–294, London, UK, 2000. Springer-Verlag.

[29] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: real-time tracking of the human body. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*, volume 19, pages 780–785, Jul 1997.

[30] C. Wu, H. Aghajan, and R. Kleihorst. Real-time human posture reconstruction in wireless smart camera networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN '08)*, pages 321–331, Washington, DC, USA, 2008. IEEE Computer Society.

[31] X. Wu and M. Liu. In-situ soil moisture sensing: measurement scheduling and estimation using compressive sensing. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks (IPSN '12)*, New York, NY, USA, 2012. ACM.

[32] T. Yan, D. Ganesan, and R. Manmatha. Distributed image search in camera sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (Sensys' 08)*, pages 155–168, New York, NY, USA, 2008. ACM.

[33] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *17th International Conference on Pattern Recognition (ICPR '04)*, volume 2, pages 28–31, Aug 2004.

A Proof that balanced projection matrices maximise detection probability

In this appendix, we will show that if the change in pixel value is given by a Gaussian distribution and if the background model changes slowly, then a balanced projection matrix maximises the detection probability for the background. For the proof, it is sufficient to consider a generic 8×8 block and one generic projection vector. Let us denote the i-th pixel value in the block at time t by $x_t^{(i)}$. We assume that the lighting change from time t to time $(t+1)$ causes the i-th pixel value at time $(t+1)$ to change to $x_t^{(i)}+\Delta_{t+1}^{(i)}$ where $\Delta_{t+1}^{(i)}\sim\mathcal{N}(\mu_{t+1},\sigma_{t+1}^2)$. We further assume that $\Delta_{t+1}^{(i)}$ is independent of all earlier pixel values. This implies that, for given $x_t^{(i)}$, the i-th pixel in the block at time $(t+1)$ is a random variable $X_{t+1}^{(i)}$ whose distribution is:

$$X_{t+1}^{(i)}\sim\mathcal{N}(x_t^{(i)}+\mu_{t+1},\sigma_{t+1}^2) \tag{12}$$

We consider a particular projection vector whose i-th element is denoted as β_i. The projection values at time t and $(t+1)$ are, respectively, $y_t=\sum_{i=1}^{64}\beta_i x_t^{(i)}$ and $y_{t+1}=\sum_{i=1}^{64}\beta_i x_{t+1}^{(i)}$. Since the change in pixel value is Gaussian distributed, it can be shown that the projection value at time $(t+1)$ is a Gaussian distributed random variable $Y_{t+1}\sim\mathcal{N}(y_t+\mu_{t+1}^p,(\sigma_{t+1}^p)^2)$ where

$$\mu_{t+1}^p \quad = \quad \left(\sum_{i=1}^{64}\beta_i\right)\mu_{t+1} \tag{13}$$

and $\sigma_{t+1}^p=64\sigma_{t+1}$ In particular, note that μ_{t+1}^p is always zero if the projection vector is balanced because the sum in the parentheses in (13) vanishes for a balanced projection vector.

Furthermore, let us assume that the background model at time t is $\mathcal{N}(\mu_t^b,(\sigma^b)_t^2)$. For a Gaussian mixture model, a projection value y_t is considered to be in the background if y_t is within 2.5 times of the standard deviation of the background model at that time. The situation for projection value at time $(t+1)$ is identical.

Our aim is to determine the probability $P(y_{t+1}\in B|y_t\in B)$, where B denotes the event that the projection value is in the background, and show that this probability is maximised by using a balanced projection vector. More specifically, we will do that by showing that $P(y_{t+1}\in B|y_t\in B)$ is maximised by choosing $\mu_{t+1}^p=0$ and with the fact that any balanced projection vector will give a $\mu_{t+1}^p=0$, our assertion can therefore be proven.

From Bayes' theorem, we have

$$P(y_{t+1}\in B|y_t\in B)=\frac{P(y_{t+1}\in B,y_t\in B)}{P(y_t\in B)} \tag{14}$$

Because we assume the change in pixel values from time t to $(t+1)$ is independent of pixel values at time t or earlier, this means the denominator is independent of μ_{t+1}^p. This implies that maximising $P(y_{t+1}\in B|y_t\in B)$ with respect to μ_{t+1}^p is identical to maximising $P(y_{t+1}\in B,y_t\in B)$.

The joint probability distribution $P(y_{t+1}\in B,y_t\in B)$ can be written as:

$$P(y_{t+1}\in B,y_t\in B)=\int_{\mu_t^b-2.5\sigma_t^b}^{\mu_t^b+2.5\sigma_t^b}\rho(y_t)P(y_{t+1}\in B|y_t)\mathrm{d}y_t \tag{15}$$

where

$$P(y_{t+1}\in B|y_t)= \int_{\mu_{t+1}^b-2.5\sigma_{t+1}^b}^{\mu_{t+1}^b+2.5\sigma_{t+1}^b}\frac{1}{\sqrt{2\pi}\sigma_{t+1}^p}e^{-\frac{(y_{t+1}-(y_t+\mu_{t+1}^p))^2}{2(\sigma_{t+1}^p)^2}}\mathrm{d}y_{t+1}, \tag{16}$$

$$\rho(y_t)=\frac{1}{\sqrt{2\pi}\sigma_t^b}e^{-\frac{(y_t-\mu_t^b)^2}{2(\sigma_t^b)^2}} \tag{17}$$

If the background learning process is stable and the learning rate is slow (e.g. 0.01), then the background model changes slowly. This means $\mu_t^b\approx\mu_{t+1}^b$. Under the assumption that $\mu_t^b=\mu_{t+1}^b$, we find the derivate of $P(y_{t+1}\in B,y_t\in B)$ with respect to μ_{t+1}^p, evaluated at $\mu_{t+1}^p=0$, is:

$$\int_{-2.5\sigma_t^b}^{2.5\sigma_t^b}\frac{1}{\sqrt{2\pi}\sigma_t^b}e^{-\frac{\hat{y}_t^2}{2(\sigma_t^b)^2}}\frac{1}{\sqrt{2\pi}\sigma_{t+1}^p}\cdot [e^{-\frac{(2.5\sigma_{t+1}^b+\hat{y}_t)^2}{2(\sigma_{t+1}^p)^2}}-e^{-\frac{(2.5\sigma_{t+1}^b-\hat{y}_t)^2}{2(\sigma_{t+1}^p)^2}}]\mathrm{d}y_t \tag{18}$$

Note that the above integrand is an odd function and the interval is symmetric about $y_t=0$, therefore the integral evaluated to zero. Hence, we prove that $P(y_{t+1}\in B|y_t\in B)$ is maximised at $\mu_{t+1}^p=0$. Therefore, when we utilise balance matrix, our method will maximise the probability of correct detection for the background.