Non-Parametric Background Detection for Video Surveillance

William Porr, Alireza Tavakkoli

Abstract.

In this paper, we propose an adaptive, non-parametric method of separating background from foreground in static camera video feed. Our algorithm processes each frame pixel-wise, and calculates a probability density function at each location using previously observed values at that location. This method makes several improvements over the traditional kernel density estimation model, accomplished through applying a dynamic learning weight to observed intensity values in the function, consequentially eradicating the large computational and memory load often associated with non-parametric techniques. In addition, we propose a novel approach to the classic background segmentation issue of "ghosting" by exploiting the spatial relationships among pixels.

1 Introduction

Accurate separation of foreground from background is essential for many video surveillance applications. The separation serves as a basis from which other, higher level surveillance applications can be conducted more efficiently. Such higher-level applications include tasks such as object tracking or, as what was done in [1], the segmentation of body parts. The general approach is to create a statistical model of the background at either each pixel or a larger surrounding region using the known values from the video feed. Then for each incoming frame, the pixel or region models are referenced using the incoming values. This "referencing" will then return a value which reflects how closely the incoming value corresponds to the background model. Those values with low probabilities of belonging to the background model are to be designated as foreground.

Unfortunately, this approach is not without its faults. Because the

background models are generally based on the pixel intensity values alone, we run into issues when those values change for reasons other than the inclusion of foreground objects. For example, Global illumination change is a common problem because it changes the values of each pixel for most, if not all pixels in the frame, often leading to many false foreground detections. Shadows are also often falsely detected as foreground, as they decrease the pixel values that they cover. However, techniques such as the use of chromasticity variables as done in [1] have been able to solve this issue. Intermittent object motion is another notorious problem with these algorithms because it produces "ghosts". These "ghosts" are falsely detected foreground regions left behind by the exit of objects that were previously considered background. Slow moving objects can be difficult to detect in certain algorithms that incorporate blind updating as well because the background model updates faster than the object itself moves. Blind updating refers to updating the probability density function, or pdf, with the incoming value regardless of whether it was detected as a foreground or background value.

In any case, much of the challenge is in creating an algorithm that is both precise and computationally efficient. Many algorithms are able to produce fantastic benchmarking results, but are too costly to be used in real time applications or embedded systems. Other algorithms are simple and rapid in their execution, but lack enough accuracy to be depended on in higher level applications. Here, we attempt to strike an effective balance between these two factors of speed and precision.

2 Related Work

If one were to invest even the smallest fraction of time investigating the field of computer vision, they would quickly become aware of the vast and diverse stores of literature addressing the problem of background subtraction. However diverse and unique these approaches may be, the majority tend to fall into a combination of a few structural categories. The most prominent of these categories being pixel based vs. region based model jurisdiction, parametric vs. non-parametric modeling, and conservative vs. blind updating.

1. Pixel-based vs. Region-based

   For most methods, background models are formed by a matrix of statistical models. Each individual model forms the background at either one particular pixel or a larger region of N x N pixels. The benefit of modeling at the pixel level, such as in [5], [7], and [8], is the potential

for sharper foreground segmentation and better capture of small foreground objects. The drawback of this approach is an increased potential for random noise affecting the background segmentation, since the spatial relationships between pixels are not taken advantage of, and a higher computational and memory cost. Region-based modeling, such as in [2] and [6], has the benefit of lower computational and memory cost, lower levels of noise, and a more cohesive capture of foreground objects (less holes, gaps, etc). The main drawbacks, as implied, are less precise foreground segmentation, poor detection of small objects, and inaccurate modeling along edges separating different background objects.

2. Parametric vs. Non-Parametric

   Parametric models, such as [5], are often structured with underlying assumptions about the arrangement of the true background, while non-parametric models, such as [1], [2], and [3], are designed to conform to the true background almost solely using observed input from the video sequence. The benefit of parametric models is that they often are less computationally demanding than their non-parametric counterparts, but the structural assumptions made can result in inaccuracies when they are not consistent with the true background. Non-parametric models have the benefit of being more flexible, however, as stated in [4], they are highly dependent on data.

3. Conservative vs. Blind Updating

   Blind updating schemes, such as [1], [3], and [8], use both observed foreground and background pixel values in order to model the background, while conservative updating schemes only make use of pixel values detected as background in order to create a background model. Blind updating schemes are better able to adapt to changes in scenery, but are typically poor in detecting slow moving foreground objects. Conservative updating schemes, as stated in [4], prevent "pollution" of the background model from foreground pixels. This comes at the cost of having to devise a method of including static foreground objects into the background after a period of time and having to create a means of initializing the background model using frames that include foreground objects as done in [2].

As stated before, the many modern algorithms contain a combination of these features along with other novel structural attributes. Maddalena et al. developed the SOBS system, which makes use of artificial neural

networks inspired by biological systems in order to create an adaptive background model. This network of neurons is able to model the background in a self-organizing manner and has many resemblances of a non-parametric system [3]. Hati et al. are able to compute a background model using their LIBS scheme by establishing an intensity range for each pixel location in the scene. This intensity range defines the upper and lower extremities for which a value can be considered background [5]. Elgammal et al. used a non-parametric approach which makes use of kernel density estimation to create a probability density function at each pixel. Sampled values at a pixel location are each assigned a weighted kernel function. These kernel functions collectively create a weighted density function from which newly observed values can be predicted as foreground or background [1].

The ViBe algorithm models the background at the pixel level, while simultaneously incorporating spatial relationships among pixels into the individual background models. An incoming value is predicted to be background if they are within a certain proximity in 3-D space of enough previously observed values at that location or its neighbor. Also, in order to update the background models and allow adaptation, ViBe randomly selects which values in the model to substitute with incoming values that are designated to be incorporated into the background model [2]. Lu in [6] developed a region based approach which creates a background model using various levels of a Gaussian pyramid. Foreground information is detected not only by observing the difference between different regions in the input frame and background model at the finest Gaussian scale, but also by the motion probability contributed by the coarser Gaussian scales. Zivkovic in [7] improves upon the widely used parametric Gaussian Mixture Model system by incorporating an on-line clustering algorithm used to separate foreground clusters from background clusters, and by allowing older information to decay exponentially. Manzanera in [8] uses a simple and effective non-linear background subtraction method which simply uses increment and decrement to update current estimates. It is popular in embedded systems because of its low computational cost.

3 Non-Parametric Background Detection

3.1 Probabilistic Function

A primary motivation in the design of our model is to improve upon the traditional Kernel Density Estimation model as used in [1]. There are certain drawbacks in the KDE model which can be improved upon signif-

icantly within the context of background subtraction for better efficiency and accuracy. One major issue with kernel density estimation is that since N kernels must be summed and averaged for computation of $P(\mathbf{x}^t|BG)$, the computational complexity of the basic method becomes $\mathrm{O}(N)$, causing a large degree of inefficiency as N becomes larger over time. In addition, when using kernel density estimation one must store each individually observed intensity in order to keep track of the kernel locations. This is obviously undesireable as it would dramatically increase the memory demand in an implementation. Also, the user only has access to individual values of $P(\mathbf{x}^t|BG)$, rather than the function as a whole since only the kernel locations are stored. Our model is inherently able to solve these problems through its incremental design.

3.1.1 Creating and Updating the Background Model

Our scheme builds a pixel-wise non-parametric density function using observed values from the various frames of a video feed. Each function represents the probability density function for one channel of one pixel in the video frame, and this model is updated as new intensities are observed at the pixel location. A model for one channel of location $\mathbf{x}$ can be represented at time t by:

$$\overset{\sim}{\theta}^{t}(\mathbf{x}) = \alpha\overset{\sim}{\theta}^{t-1}(\mathbf{x}^t) - \beta\overset{\sim}{\theta}^{t-1}(\sim \mathbf{x}^t) \tag{1}$$

such that

$$\sum_{i=0}^{D} \overset{\sim}{\theta}^{t}(\mathbf{x}_i) = 1 \tag{2}$$

where $\mathbf{x^t}$ is the most recently observed channel intensity at location $\mathbf{x}$, $\sim \mathbf{x^t}$ reflects all values in the function that are not $\mathbf{x^t}$, and D is the size of the domain of the function. α is the value by which $\overset{\sim}{\theta}^{t-1}(\mathbf{x^t})$ is updated at time t, and β is the value by which all unobserved values of the function are forgotten. In this sense, α reflects the learning rate of the function, and β the forgetting rate. For our purposes, α and β can be substituted by linear functions, such that:

$$\alpha = \frac{N^{t-1} - (\overset{\sim}{\theta}^{t-1}(\mathbf{x}^t) \cdot N^{t-1})}{(N^{t-1})^2 + N^{t-1}} \tag{3}$$

and

$$\beta = \frac{\overset{\sim}{\theta}^{t-1}(\mathbf{x}^t) \cdot N^{t-1}}{(N^{t-1})^2 + N^{t-1}} \tag{4}$$

where N^t is the number of samples in the function with relation to time t. These functions reflect decreasing values of α and β which approach 0 as $N \to \infty$. We chose these dynamic values of α and β for the computational efficiency and accuracy it provides. A decreasing learning and forgetting rate allows for the formation of a general model quickly in the beginning of the video sequence since earlier frames will have the largest influence. This rough model is then finely tuned during latter frames of the sequence using the lowered rates.

One may desire for a learning rate which approaches a constant value rather than zero as $N \to \infty$. In that case, the eq.1 could be reworked as:

$$\overset{\sim}{\theta}^{t}(\mathbf{x}) = \frac{\alpha\overset{\sim}{\theta}^{t-1}(\mathbf{x^t})}{1+\alpha} + \frac{\overset{\sim}{\theta}^{t-1}(\sim \mathbf{x^t})}{1+\alpha} \tag{5}$$

where

$$\alpha = \frac{1 + Z \cdot N^{t-1}}{N^{t-1}} \tag{6}$$

where Z is the constant α will approach as $N \to \infty$. This approach would have the benefit of increased adaptability at later frames.

In order to determine if location $\mathbf{x}$ is a background pixel, the process used in eq.1 to model the function at time t is then repeated for the other two channels of pixel $\mathbf{x}$. In order to calculate the probability of $\mathbf{x}$ being a part of the background using the three probabilistic values gathered from eq.1, we use the following equation:

$$P(BG|\mathbf{x}) = \overset{\sim}{\theta}^{t}(\mathbf{x}_B^t) \cdot \overset{\sim}{\theta}^{t}(\mathbf{x}_G^t) \cdot \overset{\sim}{\theta}^{t}(\mathbf{x}_R^t) \tag{7}$$

By multiplying these probabilities, we are able to acquire a probabilistic value which assumes a statistical independence at each channel. To decide which values are associated with the background and which are to be considered foreground, we establish a global threshold value th. Now, we determine that pixel $\mathbf{x}$ is a background pixel only if its probability as gathered from eq.7 is greater than th. Also, for the purposes of compuatational efficiency and improving end results, we adopted a parameter σ to serve as a bandwith for the probability density function. This bandwidth should be chosen in such a way that the effect of random noise in the video sequence is diminished, while avoiding the effects of oversmoothing.

3.2 Filters

Unfortunately, even with simple video samples, we run into issues due to noise. Random noise present in the clear majority of videos causes a degree of inaccuracy when updating our background model and in attempting to determine if a pixel affected by noise corresponds the background. As a result, each binary output frame is often subject to misclassified foreground pixels due to significant noise in the video sequence. This is undesirable for obvious reasons, mostly as it harms the overall precision of the algorithm. For this reason, we decided to include a simple 5x5 median window filter and 3x3 Gaussian blur in our background subtraction algorithm.

3.2.1 Processing the Binary Output Frame

After each frame is processed and a binary output frame is produced, we apply a Gaussian blur to the binary image to soften the effect of these missclassified pixels. This image is then fed into a median filter. This filter operates by filtering out misclassified foreground pixels based on the presence of foreground pixels in the surrounding neighborhood. For example, if we have a binary output frame B, and pixel B_i is detected as a foreground pixel, then pixel B_i shall remain a foreground pixel only if more than half of the pixels surrounding it within a 5 x 5 space are *also* detected as pixels associated with a foreground object. These simple solutions have shown to improve results significantly.

3.3 Global Illumination Change

As we stated before, Global Illumination Change is a characteristic problem associated with algorithms such as the one we propose here. Our method of background subtraction is unable to natively consider global changes in illumination, so we must make an exception. Also, since our algorithm fundamentally assumes that the video feed comes from a stationary source, we run into issues when a camera is shifted for whatever intentional or unintentional reason. If either condition arises, the native algorithm will adapt only slowly, which is undesirable for the vast majority of applications as it would inhibit accurate background segmentation during that period. For this reason, we simply structured the program to clear the probability density function for every channel of every pixel in the frame. It is a simple addition which saves the program from a large number of false detections, ensuring that it can run smoothly through abnormal conditions.

3.4 Ghost Detection

Ghostsin the field of computer vision, as explained in section 1, represent a notoriously difficult issue to resolve. Our algorithm can adapt over time to include foreground objects into the background model, however as was the case with global illumination changes, this adaptation is too slow to be desirable in real time applications when dealing with a falsely detected foreground region. Differentiating between true foreground objects and ghosts historically has been a difficult task to solve, yet we have devised a method to do just that.

3.4.1 The Procedure

We make use of the fact that, as stated in [2], the values of the background surrounding the ghost object tend to be similar to those values of the ghost itself. Using this information, we devised a method of comparing nearly all the values near the edge of a ghost with those background values in the same vicinity. By using this information, we are able to effectively differentiate between ghosts and true foreground objects, where we can proceed to make appropriate adjustments.

In order to make these comparisons, we first gather the silhouette of the detected foreground region in question using the binary output frame. For each pixel along the silhouette (or just a portion for improved performance, the logic still holds), we gather two nearby values from the *original* frame, one outside the boundary of the silhouette and one inside. We then subtract these two values and add the absolute value of that difference to an accumulating sum. This sum is then divided by half of the total number of samples gathered for that particular foreground object. If in fact the object is a ghost and the surrounding background region shares similar values, the average difference after processing the samples should be about zero. However, we do not always end up with such a result simply because backgrounds themselves can, for instance, have elaborate patterns and edges. This, with the addition of the inaccuracies caused by random noise, leads us to establish a threshold *near* zero in order to distinguish ghosts from static foreground objects, while still compensating for backgrounds that contain elaborate textures and edges. For example, say we have a silhouette and two arrays (O and I) of sampled values, one corresponding to values gathered outside the silhouette boundary and one corresponding to those values gathered inside the boundary. Each value was gathered according to its proximity to a silhouette pixel, meaning that pixel O_i should be within the

same neighborhood as I_i. We then take the average difference of the corresponding values between the two arrays and compare it to a designated threshold value by

$$(\sum_{k=0}^{N} O_k - I_k)/N < t \tag{8}$$

N being the size of each array and t representing the set threshold value. If this Boolean operation returns true, then we designate that foreground region as a ghost. If the operation does not return true, then we designate the region as a static foreground region, since detectable foreground objects are presumed to have no predictable correlation with the background. A visual example of this situation is given in 3.4.1.

If we determine the foreground region in question to be a static foreground object, then we do not disturb the background models at those locations. However, if we have detected the foreground region as a ghost, we will clear all the observed values of each probability density function contained within the ghost object. By doing this we discard what has become irrelevant data about foreground objects and allow the model to reform to the background, which also immediately suppresses the false foreground detection caused by the ghost.

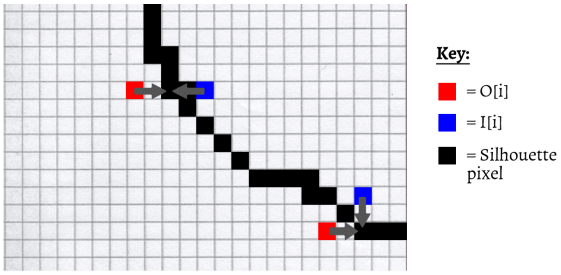

Figure 1: A visual example of how pixel samples are gathered for detecting ghosts

4 Experimental Results

4.1 Benchmarking Dataset

Though the construction of these various Background Subtraction algorithms may hold up logically, it is impossible to take into consideration all possible factors or situations. Therefore, the only true test is the real-world

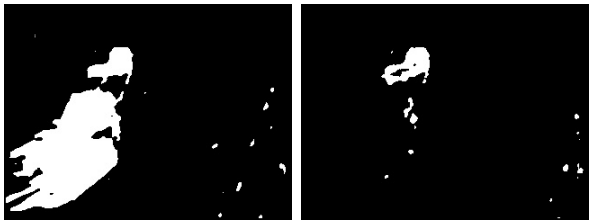

Figure 2: Before adding the Ghost module on the left, after on the right with t=5

application of the logic used. To accomplish an accurate comparison of performance between various methods, we chose to use the 2014 dataset from changedetection.net [9]. This dataset provides a variety of environmental conditions which may be experienced in real-world usage of background-subtraction programs, resulting in a useful benchmarking tool that highlights the strengths and weaknesses of the methods used. An overview of the 2014 dataset can be found here [9](include the measurements we decided to use)

4.2 Our Parameters

The parameters we used for our core algorithm are as follows. We set our bandwidth to 42.67 and $th = 0.03$, which proved to provide the speed and accuracy required. We updated each probability density function at every interval of 8 frames, striking a strong balance between speed and accuracy. Also to further increase performance, we integrated CPU multithreading technology into the implementation of out algorithm. We believed that this was an appropriate addition considering how multi-core CPUs have been widely adopted and will likely be adopted even further as time progresses [10]. Therefore, we have provided an implementation of our method which will accurately reflect how the algorithm would perform in modern applications.

4.3 Choice of Other Methods

We used three state-of-the-art algorithms to test against our program, SOBS [3], ViBe [2], and the Zipfian method [8]. The purpose of this comparison is to show how our algorithm performs relative to established methods, facilitating the process of judging our method's value in the field of computer

vision. SOBS was chosen to serve as an example of the relative upper boundary in segmentation precision possible today, which is coupled by its often slow computational times. On the other end of the spectrum, the Zipfian method was chosen as an example of the relative upper boundary in computational time. ViBe was chosen as one of the better-established methods which seeks to strike the same balance between computational speed and precision as we do in this paper.

4.4 Parameters of Other Methods and Specifications

The parameters of SOBS are $e1 = 1.0$, $e2 = 0.008$, $c1 = 1.0$, $c2 = 0.05$; any parameters not listed were left at default as given in [11]. For ViBe, the opencv C++ implementation was used to gather the results. We acquired the source code from [12] and left the parameters at their default values. The parameters of the C implementation of the ZipFian method were left at the default values set by the author who kindly provided the source code. The raw TP, FP, FN, and TN numbers were computed using the C++ comparator program provided by changedetection.net. The ViBe and Zipfian methods, as well as our method, were built from source and tested on Linux Mint 18 Cinnamon 64-Bit with Linux Kernel version 4.4.0-21-generic, while the SOBS method was tested on Windows 10 using the executable found on the author's website [11]. All methods were tested using an Intel core i5-4690k clocked at 4.3 GHz with four cores/threads and 16 GB of ram.

Category	Measurement	SOBS	Proposed Method	ViBe	ZipFian
baseline	Recall	.771	.609	.534	.648
baseline	Specificity	.998	.996	.998	.856
baseline	FPR	.001	.003	.001	.143
baseline	FNR	.228	.390	.465	.351
baseline	PWC	.902	1.989	1.954	15.278
baseline	F-Measure	.839	.709	.680	.296
baseline	Precision	.920	.868	.994	.232
Camera Jitter	Recall	.717	.594	.452	.604
Camera Jitter	Specificity	.972	.960	.998	.869
Camera Jitter	FPR	.027	.039	.011	.130
Camera Jitter	FNR	.282	.405	.547	.359
Camera Jitter	PWC	3.737	5.323	3.311	13.951
Camera Jitter	F-Measure	.616	.492	.521	.267
Camera Jitter	Precision	.545	.429	.638	.174
Dynamic Background	Recall	.726	.636	.443	.524

Dynamic Background	Specificity	.985	.979	.997	.966
Dynamic Background	FPR	.014	.020	-	.033
Dynamic Background	FNR	.273	.363	.556	.475
Dynamic Background	PWC	1.643	2.448	.822	3.840
Dynamic Background	F-Measure	.528	.443	.504	.215
Dynamic Background	Precision	.467	.402	.671	.153
Intermittent Object Motion	Recall	.549	.365	.263	.357
Intermittent Object Motion	Specificity	.901	.984	.982	.946
Intermittent Object Motion	FPR	.098	.015	-	.053
Intermittent Object Motion	FNR	.450	.634	.736	.642
Intermittent Object Motion	PWC	11.012	6.136	6.905	9.638
Intermittent Object Motion	F-Measure	.495	.379	.330	.272
Intermittent Object Motion	Precision	.554	.691	.674	.269
Shadow	Recall	.729	.665	.545	.650
Shadow	Specificity	.988	.975	.994	.932
Shadow	FPR	.011	.024	.005	.067
Shadow	FNR	.270	.334	.454	.349
Shadow	PWC	2.234	3.737	2.399	7.912
Shadow	F-Measure	.731	.588	.659	.407
Shadow	Precision	.740	.581	.874	.305
Thermal	Recall	.482	.439	.296	.639
Thermal	Specificity	.996	.990	.999	.882
Thermal	FPR	.003	.009	-	.117
Thermal	FNR	.517	.560	.703	.360
Thermal	PWC	2.641	5.071	4.591	13.570
Thermal	F-Measure	.594	.519	.434	.391
Thermal	Precision	.862	.699	.984	.317
Bad Weather	Recall	.569	.643	.434	.787
Bad Weather	Specificity	.997	.993	.996	.855
Bad Weather	FPR	-	.006	-	.144
Bad Weather	FNR	.430	.356	.565	.212
Bad Weather	PWC	.872	1.277	1.287	14.630
Bad Weather	F-Measure	.666	.609	.555	.236
Bad Weather	Precision	.835	.618	.857	.166
Low Framerate	Recall	.550	.419	.263	.486
Low Framerate	Specificity	.955	.988	.982	.928
Low Framerate	FPR	.044	.011	-	.071

Low Framerate	FNR	.449	.580	.736	.513
Low Framerate	PWC	5.779	2.873	6.905	8.238
Low Framerate	F-Measure	.472	.379	.330	.248
Low Framerate	Precision	.548	.451	.674	.184
Night Videos	Recall	.603	.491	.283	.638
Night Videos	Specificity	.958	.967	.993	.862
Night Videos	FPR	.041	.032	.006	.137
Night Videos	FNR	.396	.508	.716	.361
Night Videos	PWC	4.944	4.326	2.147	14.225
Night Videos	F-Measure	.363	.308	.331	.179
Night Videos	Precision	.301	.267	.483	.115
PTZ	Recall	.699	.582	.313	.483
PTZ	Specificity	.682	.866	.904	.722
PTZ	FPR	.317	.133	.095	.277
PTZ	FNR	.300	.417	.686	.516
PTZ	PWC	31.780	13.681	10.152	28.103
PTZ	F-Measure	.040	.195	.089	.033
PTZ	Precision	.021	.170	.059	.018
Turbulence	Recall	.631	.672	.578	.696
Turbulence	Specificity	.994	.971	.999	.898
Turbulence	FPR	.005	.028	-	.101
Turbulence	FNR	.368	.327	.421	.303
Turbulence	PWC	.750	3.013	.269	10.248
Turbulence	F-Measure	.463	.234	.674	.062
Turbulence	Precision	.438	.163	.822	.034

Method	SOBS	Proposed Method (four threads)
FPS:	16.19	545.46
Method	Proposed Method (single-threaded)	ViBe
FPS:	169.82	127.66
Method	ZipFian (grayscale)	
FPS:	816.58	

4.5 Final Thoughts

From the results we gathered in our experiment, it is evident that there are certain categories which some methods execute better than others, and that no one solution completely dominates the other three. In particular, it can be seen through the results that the ViBe implementation excels

at datasets which incorporate noise, small camera displacement, or dynamic background movements. This is a result of ViBe's use of spatial relationships in updating pixel models, effectively suppressing noise and dynamic background movements such as trees or waves. For our method, we see strong results in the baseline category, which is meant to represent the more typical surveillance scenarios, and in the low framerate category. Our success in the baseline category shows the probability density functions' ability to increasingly approach the true density when left relatively undisturbed. The results in the latter mentioned category can be attributed to our non-parametric model and lack of an initialization phase, such as that used in [2] and [3]. This allows our model to adapt over time and prevent initial misclassifications from persisting throughout execution. These missclassifications would be accentuated by a low framerate sequence where foreground objects are likely to appear in more areas of the scene due to the increased frame-interval time. It is evident that many categories, aside from the few mentioned above, show similar outcomes between our algorithm and ViBe. Also, due to our increment-decrement approach, the computational complexity of the proposed method is O(1), leading to an extremely efficient implementation. Consequentially, though when multithreaded the proposed method is far more efficient than ViBe or SOBS, in single threaded applications it is still able to outperform ViBe by 42 frames per second and SOBS by 154 frames per second, while still proivding sufficient accuracy most surveillance senarios. Based on these results, it seems that the proposed method would be useful in typical surveillance applications, especially where high resolution and high frame-rate surveillance videos are needed. Additionally, the efficiency and accuracy of this solution would also prove useful in situations where low power and low memory embedded systems are used.

Additionally, though it was not included during benchmarking, the ghost detection method has proven its potential. The best use of the method would likely be in applying it only to stationary objects. This, as opposed to simply applying it to all foreground objects, would significantly reduce the computational cost of its addition.

5 Examples

Here are example frames for each method in each category, in the order of GroundTruth, SOBS, Our Method, ViBe, and ZipFian

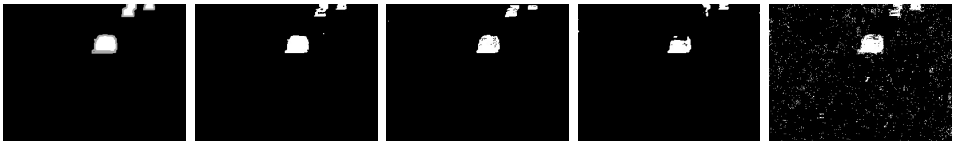

Figure 3: Baseline

Figure 4: Camera Jitter

Figure 5: Dynamic Background

Figure 6: Intermittent Object Motion

Figure 7: Shadow

Figure 8: Thermal

Figure 9: Bad Weather

Figure 10: Night Videos

Figure 11: Low Framerate

Figure 12: PTZ

Figure 13: Turbulence

6 Pseudocode

Below is some example pseudocode of the main algorithm and the algorithm for ghost detection.

6.1 Main Algorithm

for ($count = 0; count <$ totalFrames; $count$+=1){
  F_t = Next frame in sequence
  B_t = Black image
  for ($i = 0; i < (F_t.\text{pixels} \cdot 3); i$+=3){
    if (!First frame){
      $P(x^t|BG) = \widetilde{\Theta}_B(F_t[i]/\alpha) \cdot \widetilde{\Theta}_G(F_t[i+1]/\alpha)$
        $\cdot\widetilde{\Theta}_R(F_t[i+2]/\alpha)$
      if ($P(x^t|BG) < \emptyset$)
        B_t[i / 3] = 255
    }
    if ((frameNumber %8) = 0){
      $\Theta_B(F_t[i]/\alpha)$+=1
      $\Theta_G(F_t[i+1]/\alpha)$+=1
      $\Theta_R(F_t[i+2]/\alpha)$+=1
    }
  }

  gaussBlur(B_t)
  medianFilter(B_t)
  DISPLAY F_t
  DISPLAY B_t
}

6.2 Ghost Detection Algorithm

for ($count = 0; count <$ totalFrames; $count$+=1){
  F_t = video frame
  B_t = processed binary image
  $N_r = F_t$.rows
  $N_c = F_t$.columns
  Sil = findSilhouettes(B_t)

  for ($i = 0; i < numberOfSilhouettes; i$+=1){
    $Sum = 0$
    for ($inc = 0; inc < sizeOfSilhouette; inc$+=1){
      $Pt = Sil$[i][inc]

$xOffset = 2$
$yOffset = 0$
$Pt_1 = F_t[(Pt.y \cdot N_c \cdot 3) + ((Pt.x - 2) \cdot 3)]$
$Pt_1 \mathrel{+}= F_t[(Pt.y \cdot N_c \cdot 3) + ((Pt.x - 2) \cdot 3) + 1]$
$Pt_1 \mathrel{+}= F_t[(Pt.y \cdot N_c \cdot 3) + ((Pt.x - 2) \cdot 3) + 2]$
$Pt_1b = B_t[(Pt.y \cdot N_c) + (Pt.x + 2)]$

if $(B_t[(Pt.y \cdot N_c) + (Pt.x + xOffset)] = Pt_1b)\{$
$xOffset = 0$
$yOffset = 2$
if $(B_t[((Pt.y + yOffset) \cdot N_c) + Pt.x] = Pt_1b)\{$
$yOffset = -2$
if $(B_t[((Pt.y + yOffset) \cdot N_c) + Pt.x] = Pt_1b)$
continue loop
}
}
$Pt_2 = F_t[((Pt.y + yOffset) \cdot N_c \cdot 3) + ((Pt.x + xOffset) * 3)]$
$Pt_2 \mathrel{+}= F_t[((Pt.y + yOffset) \cdot N_c \cdot 3) + ((Pt.x + xOffset) * 3) + 1]$
$Pt_2 \mathrel{+}= F_t[((Pt.y + yOffset) \cdot N_c \cdot 3) + ((Pt.x + xOffset) * 3) + 2]$
$Diff = |Pt_1 - Pt_2|$
$Sum \mathrel{+}= Diff$
}
$Avg = Sum/sizeOfSilhouette$

if $(Avg < t)\{$
$Top = Bottom = Left = Right = 0$

for $(inc = 0; inc < sizeOfSilhouette; inc\mathrel{+}=1)\{$
$Pt = Sil$[i][inc]
if $(Pt.y < Top)$
$Top = Pt.y$
if $(Pt.y > Bottom)$
$Bottom = Pt.y$
if $(Pt.x < Left)$
$Left = Pt.x$
if $(Pt.x > Right)$
$Right = Pt.x$
}
for $(inc = Top; inc < Bottom; inc\mathrel{+}=1)\{$
for $(cnt = Left; cnt < Right; cnt\mathrel{+}=1)\{$

```
            if (B_t[(inc · N_c) + cnt] = 255){
               for (N = 0; N < (256/α); N+=1){
                  Θ_B(N) = 0
                  Θ_G(N) = 0
                  Θ_R(N) = 0
               }
               Θ_B(F_t[(inc · N_c · 3) + (cnt · 3)]/α)+=1
               Θ_G(F_t[(inc · N_c · 3) + (cnt · 3) + 1]/α)+=1
               Θ_R(F_t[(inc · N_c · 3) + (cnt · 3) + 2]/α)+=1
            }
         }
      }
   }
```

7 Conclusion

In this paper, we presented a non-parametric solution to background subtraction in video surveillance applications. Our method operates on the pixel level using probability density functions built from observed values in the video feed to create a background model. These models then naturally develop to represent the true background of the video. We also developed a novel approach to solving the classic issue of ghosting. This was achieved by taking the difference between the pixel values at the edge of a detected foreground object and the pixel values of the background surrounding the object to distinguish true foreground objects from ghosts. Our algorithm has proven to be computationally efficient while still maintaining a level of precision comparable to established methods. This, when coupled with the method's extreme multithreading potential, makes it a viable option for modern surveillance systems.

References

[1] Elgammal, A., Duraiswami, R., Harwood, D., and Davis, L. S. (2002). Background and Foreground Modeling Using Nonparametric Kernel Density Estimation for Visual Surveillance. *Proceedings of the IEEE*, volume 90(issue 7), pp. 1151-1163, Jul. 2002.

[2] Barnich, O., Droogenbroeck, M. V. (2010). ViBe: A Universal Background Subtraction Algorithm for Video Sequences. *IEEE Transactions on Image Processing*, volume 20(issue 6), pp. 1709-1724, Dec. 2010.

[3] Maddalena, L., Petrosino, A. (2008). A Self-Organizing Approach to Background Subtraction for Visual Surveillance Applications. *IEEE Transactions on Image Processing*, volume 17(issue 7), pp. 1168-1177, May 2008.

[4] Piccardi, M. (2004). Background subtraction techniques: a review. *IEEE International Conference on Systems, Man and Cybernetics*. The Hague, Netherlands, Mar. 2005.

[5] Hati, K. K., Sa, P. K., and Majhi, B. (2013). Intensity Range Based Background Subtraction for Effective Object Detection. *IEEE Signal Processing Letters*, volume 20(issue 8), pp. 759-762, May 2013.

[6] Lu, X. (2014). A Multiscale Spatio-Temporal Bakcground Model For Motion Detection. *IEEE International Conference on Image Processing (ICIP)*. Paris, France, Oct. 2014.

[7] Zivkovic, Z. (2004). Improved Adaptive Gaussian Mixture Model for Background Subtraction. *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)*. Cambridge, UK, Aug. 2004.

[8] Manzanera A. (2007)$\Sigma - \Delta$ Background Subtraction and the Zipf Law. In: Rueda L., Mery D., Kittler J. (eds) Progress in Pattern Recognition, Image Analysis and Applications. CIARP 2007. Lecture Notes in Computer Science, vol 4756. Springer, Berlin, Heidelberg

[9] Y. Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar, CDnet 2014: An Expanded Change Detection Benchmark Dataset, in Proc. IEEE Workshop on Change Detection (CDW-2014) at CVPR-2014, pp. 387-394. 2014

[10] Min, J. (2014, 11 Dec.). *Multicore Goes Mainstream*. Retrieved from http://embedded-computing.com/articles/multicore-goes-mainstream/

[11] SOBS Executable for Windows: http://www.na.icar.cnr.it/ maddalena.l/MODLab/SoftwareSOBS.html

[12] ViBe Source Code, Original implementation: http://orbi.ulg.ac.be/handle/2268/145853