Proceedings of the 5th WSEAS Int. Conf. on SIGNAL, SPEECH and IMAGE PROCESSING, Corfu, Greece, August 17-19, 2005 (pp282-287)

GW4: An FPGA-driven Image Segmentation Algorithm

KOFI APPIAH† ANDREW HUNTER† TINO KLUGE‡
Department of Computing and Informatics†
University of Lincoln
Lincoln, LN6 7TS
UK
OCIAM, Mathematical Institute‡
University of Oxford
Oxford, OX1 3LB
UK
http://facs.lincoln.ac.uk/Research/Vision

Abstract: - We describe "GW4," an efficient video segmentation algorithm designed for FPGA implementation. The algorithm detects moving foreground objects against a multimodal background; it is motivated by two well-known adaptive background differencing algorithms, Grimson's algorithm and W^4. GW4 is designed specifically for implementation on reconfigurable FPGA hardware, avoiding the use of floating point numbers and transcendental operations, and operates at real-time frame rates on 640x480 video streams. We present experimental results indicating processing speeds, and superior segmentation performance to Grimson's algorithm.

Key-Words: - FPGA, Multimodal background, Real-time processing, Reconfigurable hardware

1 Introduction

Video segmentation algorithms process large amount of data, and are consequently processor and memory hungry [11, 15]. Typically they can be broke down into three major stages [3]: early processing, implemented by local pixel-level functions; intermediate processing, which includes segmentation, motion estimation and feature extraction; late processing, including interpretation and using statistical and artificial intelligence algorithms. Typically algorithmic sophistication is concentrated in the later stages, but processing demands are dominated by the early stages. Limitations on processing power force us to use extremely simple algorithms for early processing, limiting performance.

This article demonstrates how real-time early digital vision can be accomplished with the use of data and instruction parallelism. Our approach spans the early and intermediate levels described above. Most image segmentation algorithms are computationally expensive and require significant storage space; however, they are also often inherently parallelisable. Field Programmable Gate Array (FPGA) systems are ideal for the implementation of such algorithms, providing that algorithms are designed with the limitations of FPGA in mind (in particular, avoidance of floating point arithmetic is recommended). Modern FPGAs provide a very appealing platform for rapid, low-cost development of specialized algorithms, due to their reconfigurable nature, as opposed to older Application Specific Integrated Circuit (ASIC) designs, which have a very long and error-prone design cycle [1].

This article presents part of a vision system for monitoring suspicious human activities in a risk prone environment. Today's technology makes it possible for a single human operator to potentially monitor multiple cameras relaying images from sites like large industrial parks and residential areas separated by great distances. The increase in numbers of these cameras makes it very hard for the operator to successfully identify behaviour of interest, leading to a research interest in automated monitoring [5]. A number of algorithms for segmentation of moving objects have already been developed, and successfully implemented in software, at least for individual video streams at low frame rates and resolutions. Very few of these algorithms have been incorporated into today's video surveillance systems, partly due to computational complexity, cost and lack of real-time capability. This makes the development of such algorithms on specialized hardware timely.

Multimodal background differencing segmentation algorithms are practical, reasonably fast and can handle some typical problems, such as camera jitter, moving foliage, water and lighting changes. They require a significant amount of floating point processing, and thus when implemented in software running on general-purpose computers are limited to low frame rates and small frame sizes. They typically absorb 80% to 90% of the entire processing time, which makes them unattractive for real-time purposes.

We present here a new multimodal background differencing segmentation algorithm, which is very simple, robust and can easily be implemented in computer hardware with maximum efficiency in terms of speed and hardware area. Our algorithm is a hybrid of two robust and well-known image segmentation algorithms (Grimson's, and W^4), which illustrates how simple algorithms can be designed for efficient FPGA implementation.

2 Previous Work

The first stage in processing for many video applications is the segmentation of (usually) moving objects. Where the camera is stationary, a natural approach is to model the background and detect foreground objects by differencing the current frame with the background. A wide and increasing variety of techniques for background modelling have been described; a good comparison is given by Gutchess *et al* [7].

The most popular method is unimodal background modelling, in which a single value is used to represent a pixel, which has been widely used due to its relatively low computational cost and memory requirements [8, 13]. This technique gives poor results when used in modelling non-stationary background scenarios like waving trees, rain and snow. A more powerful alternative is to use a multimodal background representation, the most common variant of which is a mixture of Gaussians [6, 12]. However, the computational demands make such techniques unpopular for real-time purposes; there are also disadvantages in multimodal techniques [6, 12, 13] including the *blending effect*, where a pixel attains an intensity value which has never occurred at that position (a side-effect of the smoothing used in these techniques). Other techniques rely heavily on the assumption that the most frequent intensity value during the training period represents the background. This assumption may well be false, causing the output to have a large error level.

2.1 Grimson's Algorithm

Grimson *et al* [12] introduced a multimodal approach, modelling the values of each pixel as a mixture of Gaussians. The background is modelled with the most persistent intensity values. The algorithm has two variants, colour and gray-scale: in this paper, we concentrate on the gray-scale version. The probability of observing the current pixel value is given as:

$$. (. _t) = \sum_{i=1}^{k} . _{i,t} . (. _t, \propto_{i,t}, \sum_{i,t}) \quad (1)$$

Where $\propto_{i,t}$, $\Sigma_{i,t}$ and $. _{i,t}$ are the respective mean, variance and weight parameters of the i^{th} Gaussian component of pixel . at time t, and . is a Gaussian probability density function

$$. (. _t, \propto_{i,t}, \sum_{i,t}) = \frac{1}{\sqrt{2\pi \sum_{i,t}}} e^{\frac{(. _t - \propto_{i,t})^2}{2\sum_{i,t}}} \quad (2)$$

A new pixel value is generally consistent with one of the major components of the mixture model and used to update the model. For every new pixel value, $. _t$, a check is conducted to match it to one of the K Gaussian distributions. A match is found when $. _t$ is within 2.5 standard deviations of a distribution. If none of the K distributions match $. _t$, the least weighed distribution is replaced with a new distribution having $. _t$ as mean, high variance and very low weight. The weights are updated as follows:

$$. _{i,t} = . _{i,t-1} + \alpha(m_{i,t} - . _{i,t-1}) \quad (3)$$

where α is the learning rate and

$$m_{i,t} = \begin{cases} 1 \text{ if there is a match} \\ 0 \text{ otherwise} \end{cases} \quad (4)$$

$1/\alpha$ defines the time constant which determines the speed at which the distribution's parameters change. Only the matched distribution will have its mean and variance updated, using the equations:

$$\propto_t = \propto_{t-1} - . (. _t - \propto_t) \quad (5)$$

$$\sum_t = (1 - .) \sum_{t-1} + . (. _t - \propto_t)^T * (. _t - \propto_t) \quad (6)$$

$$. = \alpha . (. _t \mid \propto_t, \sigma_t) \quad (7)$$

The first *B* distributions (ordered by $. _k$) are used as a model of the background, where

$$B = \arg_b \min(\sum_{k=1}^{b} . _k > T) \quad (8)$$

The threshold T is a measure of the minimum portion of the data that should be accounted for by the background.

2.1 The W^4 Algorithm

Haritaoglu *et al* [8] introduced the W^4 algorithm, which uses a single distribution with three integer values to model the background. Their background model requires manual initialisation; the three parameters (Maximum, Minimum and maximum inter-frame difference values) are acquired over a period of time (a few seconds) when there is no activity in the scene.

After the initialisation period, each pixel is classified as either a background or a foreground pixel using the background model. Given the maximum (M), minimum (m) and the largest inter-frame absolute difference (D) of the images collected over the initialisation period, a new pixel *x* from an image sequence I_t is a foreground pixel if:

$$| m(x) - I_t(x) | > D(x) \quad (9)$$

or

$$| M(x) - I_t(x) | > D(x) \quad (10)$$

The W^4 algorithm is attractive in maintaining all values without floating point numbers, but its unimodal and cannot

model multi-modal background. If the background model is not updated over time to incorporate newly introduced background objects there will be accumulated errors. In contrast, Grimson's algorithm [12] is robust to outdoor environments where lighting intensity can suddenly change and handles multi-modal backgrounds without manual initialisation. Unfortunately, the use of floating-point numbers in all its update parameters makes it computationally expensive, and unsuitable for hardware implementation [2].

3 The GW4 Algorithm

We present here a novel hybrid image segmentation algorithm, GW4, that combines the attractive features of Grimson's algorithm and W^4 [8, 12], with appropriate modifications to improve segmentation of the foreground image, and to allow an efficient implementation on a reconfigurable hardware platform, Field Programmable Gate Array (FPGA).

Following Grimson [12], we maintain a number of clusters, each with weight $\cdot_k$, where $1 = k = K$, for K clusters. Rather than modelling a Gaussian distribution, we maintain a model with a central value, c_k. We use an implied range, $[c_k - 15, c_k + 15]$, rather than explicitly modelling a range as in W^4 [8]. The choice of *30* as the width of the clusters was based on the maximum inter-frame absolute difference obtained for some randomly selected test data (outdoor and indoor scenes) using the algorithm presented in [8]. The weights of all the clusters are initialised to 10, and the total weight remains constant.

A pixel $\cdot = I(i, j)$ from an image I is said to match a cluster, k, if $\cdot = c_k - 15$ and $\cdot = c_k + 15$. The highest weight matching cluster is updated, if and only if its weight does not exceed 50% of the total weight of all K clusters (i.e. $\cdot_k < 15$, given K=3). The update is as follows:

$$\cdot_{i,t} = \begin{cases} \cdot_{k,t-1} + (K-1) & \text{for the winning cluster} \\ \cdot_{k,t-1} - 1 & \text{otherwise} \end{cases} \quad (11)$$

If no matching cluster is found, then the least weighted cluster's central value, c_k, is replaced with X; its weight stays the same. The way we construct and maintain clusters make our approach free from the blending effect. This is because for every cluster, the central value c_k represents an intensity value which has occurred at that pixel location.

The K distributions are ordered by weight, with the most likely background distribution on top. Similar to [12], the first B clusters are chosen as the background model, where

$$B = \arg_b \min\left(\sum_{k=1}^{b} \cdot_k > T\right) \quad (12)$$

The threshold T is a measure of the minimum portion of the data that should be accounted for by the background. The choice of T is very important, as a small T usually models a unimodal background while a higher T models a multi-modal background.

We classify a pixel as foreground based on the following two conditions:

1. If the intensity value of the pixel matches none of the K clusters.
2. If the intensity value is assigned to the same cluster for two successive frames, and the intensity values *X(t)* and *X(t-1)* are both outside the 40% mid-range.

The second condition is needed to enable us to detect targets with low contrast against the background, while maintaining the concept of multi-modal backgrounds. A typical example is a moving object with gray-scale intensity close to that of the background, which would be classified as background in [12]. This requires the maintenance of an extra frame, with values representing the recently processed background intensities, but the memory requirement is not excessive due to the use of integer values in our overall computations.

The resulting foreground image is cleaned up by morphological opening using a 3 x 3 structuring element; we use the same procedure with Grimson's algorithm.

4 Hardware Implementation

Our segmentation algorithm, described in section 3, has the advantage of being computationally simple, making it suitable for hardware implementation. The mixture of Gaussian models maintained for each pixel in [12] poses a large computational and storage problem. Jiang *et al* [9] report an implementation on an SGI 02 with a R10000 processor, which can process only 11-13 frames per second with a frame size of 160 x 120. The pixel-level processing used in our GW4 algorithm makes it a good candidate for parallel and pipeline processing.

Efficient hardware implementation of any Digital Signal Processing (DSP) algorithm can be achieved in two distinct and important domains: speed and hardware area. Many DSP implementations tend to focus on one of these and ignore the other, either partially or totally. Typical general purpose-processors run at a speed of 2 to 3 GHz as compared to high-end reconfigurable computers like FPGA, which can run at a maximum speed of 200 to 500 MHz but can support parallel execution. DSP processors perform better than FPGAs when the algorithm relies heavily on floating-point numbers, since the hardware area consumed by floating point accumulators limits the parallel nature of FPGAs [2].

Real-time image processing on FPGA has three major constraints [10]: timing, bandwidth and resource constraints. These constraints have been dealt with in our implementation with the use of fixed-point numbers from the onset of the

design. The reduced hardware area makes it possible to meet the timing constraints and hence real-time processing needs. All morphological operations are conducted on BlockRAM, as a means of reducing the bandwidth constraints. In addition to the use of fixed-point numbers, our implementation minimizes resource requirements. As compared to [12], where the weight, variance and mean of each pixel is maintained for all K distributions, our approach only maintains the central value and weight for each pixel, thus reducing the storage requirement by a factor of 3*K* for each pixel. Other implementations tend to convert floating-point based algorithms into fixed-point [1] as a means of making hardware implementation feasible, but without redesign of the algorithm. The end result is accumulated error. In contrast, our algorithm is designed "from the ground up" to use fixed point arithmetic.

Our design is a fully parallel and pipelined architecture based on FPGA, which reads, processes and store a pixel every clock cycle.

There are six distinct blocks running in parallel with each other. These are:

Input Block; This block reads pixels from the camera in 24-bit RGB format at PAL frame rate (25 fps) for processing. A special mechanism had to be introduced to deal with the high disparity in frequency of the design and the camera. This block iterates several times until the expected pixel value is transmitted from the camera. Thus in effect this block runs at a maximum frequency of 25Hz.

Pixel Processing Block; This is a 6 stage pipelined block. The first stage identifies the pixel read by the input block. The memory address corresponding to the storage location of its background parameters is computed. The stored parameters are then retrieved from memory in the second pipelined stage. The third pipeline stage involves the conversion of the 24-bit RGB pixel value from the camera into 8-bit gray-scale intensity. To reduce computational cost, the well-known Craig's formula for converting RGB to Gray-scale,

$$Y = 0.3 * R + 0.59 * G + 0.11 * B \qquad (13)$$

has been modified as follows

$$Y = 0.25 * R + 0.50 * G + 0.25 * B \qquad (14)$$

This can be accomplished in a single clock-cycle with two-hardware adders and two shift operations. The fourth pipeline stage is used for pixel classification and the last two stages are used for updating the parameters of that pixel stored in external memory. The nature of the external RAM calls for two blocks of RAM to be used in parallel. Thus while the background data is been read from one block the updated data is written to the other. These blocks are then interchanged after processing a full frame.

Erosion Block; This block is use for morphological erosion. The binary foreground extracted in the Pixel Processing block is stored on a dual-port BlockRAM for erosion.

Dilation Block; This block is use for morphological dilation. The binary foreground obtained after erosion in the erosion block is further dilated and stored in another dual-port BlockRAM for the external VGA.

Pixel Output Block; This block makes data available to the VGA at its refresh rate. This is the foreground obtained after dilation.

Memory Control Block; This controls the RAM block for reading and writing. Since the external RAM is not dual-port and we need to read from and write to RAM every clock cycle, we maintain two RAM blocks, which are swapped after processing each frame.

The development of these blocks has been accomplished using Celoxica's DK3 design suite and Xilinx ISE 7.1i place and route (PAR) tool. The hardware platform is composed of a Xilinx Virtex II XC2V6000 FPGA, with equivalent of 6 million logic gates and 2,592KB of dual-port selectRAM [14]. This FPGA has been packaged with 4 banks (36-bit addressable) of external ZBT SRAM totalling 32Mbytes on the RC300 [4]. Table 1 is a summary of the resource utilization of the hardware implementation, using device *xc2v6000*, and package ff1152 and speed grade 6.

Resource	**Total Used**	**Per.**
Flip Flops	1,479 out of 67,584	2%
4 input LUTs	3,200 out of 67,584	4%
Block RAMs	57 out of 144	39%
bonded IOBs	335 out of 824	40%
GCLKs	4 out of 16	25%
DCMs	1 out of 12	8%
Occupied Slices	2,022 out of 33,792	5%

Table 1: Resource utilization on the implementation

5 Experimental Results

To evaluate the performance of our approach (GW4) against Grimson's algorithm [12] we use four video sequences, two each from outdoor and indoor scenes. The experiments were conducted using *K=3* for both algorithms. We have constructed reference standard segmentations on these sequences by using manually marked frames; results of the algorithms are compared to this reference standard. Fig. 1 shows some sample frames of the sequences and their corresponding manually marked frames.

We report pixel-wise errors against the reference standard, in terms of true positive, true negative, false negative and false positive pixels. Table 2 shows the sensitivity (SENS.), specificity (SPEC.) and the positive predictive values (PPV) of Grimson and GW4. Each of the image sequences has thirty-five frames, but the first three frames are ignored in our evaluation as they are used by the algorithms to calibrate their parameters.

Fig. 1: Sample images with manually mark-out frames.

Scene	Grimson (%)		
	SENS.	SPEC.	PPV
Indoor1	72.14	99.82	97.92
Outdoor1	70.35	99.96	98.93
Indoor2	83.76	92.69	53.57
Outdoor2	66.13	99.81	98.55
	GW4 (%)		
Indoor1	75.77	99.73	97.08
Outdoor1	74.06	99.95	98.84
Indoor2	86.39	90.38	47.46
Outdoor2	70.47	99.76	98.31

Table 2: Sensitivity, Specificity and Positive Predictive Values (PPV) of Grimson and GW4.

The evaluation parameters used are defined as follows:

$$Sensitivity(SENS.) = \frac{TP}{TP + FN}$$

$$Specificity(SPEC.) = \frac{TN}{TN + FP}$$

$$PPV = \frac{TP}{TP + FP}$$

Where *TP* is True Positive, *FN* is False Negative, *TN* is True Negative and *FP* is False Positive.

Table 2 clearly indicates the superiority of our algorithm in detecting foreground pixels (higher sensitivity than Grimson) and hence suitable for our target application. The algorithm does sometimes produce more false positive errors; this is a side-effect of the sensitivity of the model in detecting moving targets with low contrast against the background, which may lead to detection of the shadows of moving targets which Grimson's algorithm would ignore. Nonetheless, overall the error rate is lower than Grimson's [12].

To show that this result is statistically significant we combine the results of all four sequences, and consider the total number of error pixels (both false negative and false positive). A chi-square test is then conducted to test the null-hypothesis that *both algorithms have equal error rates*; see table 3. The chi-square value is defined by

$$\chi^2 = \sum \frac{(observed - \exp ected)^2}{\exp ected} \qquad (15)$$

	False pos	False neg	overall
Grimson better	99	0	24
GW4 better	29	128	107
χ^2	38.3	128	52.6

Table 3: Chi-square significance test

As it can be seen, the χ^2 value exceeds the 99% significance threshold of *6.63* for false positives, false negatives and overall errors, indicating a statistically significant performance improvement in GW4. Fig. 2 shows some sample output images from the two algorithms.

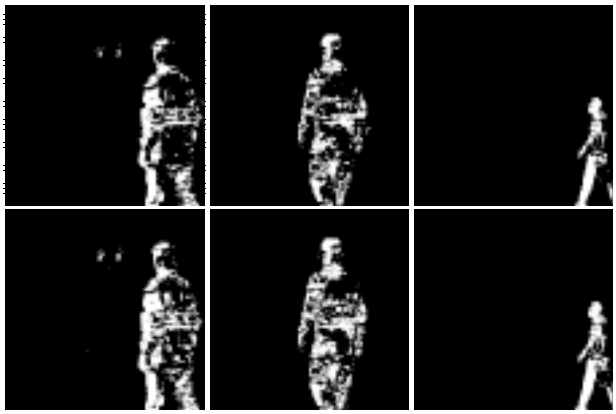

Fig. 2: Sample output of the two algorithms.
Top: Grimson's, Bottom: GW4

It is worth noting that Grimson's algorithm [12] although more sophisticated than ours, failed to perform better with the test data presented here, for *K=3*. This is due to the additional condition (condition 2) we use in extracting foreground pixels, thereby resolving the foreground aperture problem. From fig. 2, it becomes clear where Grimson's algorithm fails to perform better than GW4 due to foreground aperture. Again the low specificity of GW4 as compared to Grimson's shows GW4's sensitivity in detecting targets with low contrast to the background as well as reflections from moving objects. Clearly any form of error is undesirable. However, in our target application FP errors of the type reported are more acceptable than FN errors, as subsystem tracking stages can discard distracters such as shadows.

Timing analysis generated by the Place and Route (PAR) tool shows that the design can run at a maximum speed of 39.72ns, meaning every stage in the design can be clocked at 25.17MHz. Hence for a standard frame size of 640x480, the design can process at least 80fps (ignoring access to external RAM), when the pipeline is full. Comparing to real-time

application requirement of 30fps, the implemented design, as it stands, meets the real-timing requirement. The efficient resource utilization of our design makes it possible to add new image processing functions, like object tracking and action interpretation to the system.

6 Conclusion

In this paper we have shown a real-time adaptive background scene modelling and maintenance technique suitable for tracking people in both indoor and outdoor environments, implemented as a System-on-Chip (SoC) using FPGA technology.

Instead of using a complicated already existing background subtraction technique, our algorithm is a hybrid version of the W^4 [8] and Grimson's [12] techniques, with some modification to enhance the sensitivity in detecting targets with low contrast against the background. The system learns and models the background scene over time to detect foreground objects, in the presence of multimodal background objects like tree branches. The algorithm has been implemented in Handel-C and runs on Xilinx Virtex II FPGA. Currently, for an image size of 640x480, the system operates at real-time (30fps). This performance level cannot be easily reached without parallel processing.

References:

[1] AccelChip, Comparison of Methods for Implementing DSP Algorithms, *SoC Central*, 2003

[2] Elham Ashari and Richard Hornsey, FPGA implementation of real-time adaptive image thresholding, *SPIE-International Society for Optical Engineering*, Dec. 2004.

[3] J. Batlle, J. Martin, P. Ridao and J. Amat, A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing, *Elsevier Science Ltd.*, 2002

[4] Celoxica, Video and Imaging solution, *http://www.celoxica.com*, 2004.

[5] M. Ekinci and E. Gedikli, Real Time Background model initialization and maintenance for video surveillance, *IJCI Proceedings of Intl. XII.Turkish Symposium on Artificial Intelligence and Neural Networks*, 2003

[6] Ahmed Elgammel, David Harwood, and Larry Davis, Non-parametric Model for Background Subtraction, *Proceedings of the 6th European Conference on Computer Vision*, Dublin, Ireland, 2000

[7] D. Gutchess, M. Trajkovic, E. Cohen-Solal, D. Lyons and K. Jain, A Background Model Initialization Algorithm for video Surveillance, *IEEE, International Conference on Computer Vision*, 2001.

[8] I. Haritaoglu, D. Harwood and L. Davis, W^4: Who? When? Where? What? A real time system for detecting and tracking people, *IEEE Third International Conference on Automatic Face and Gesture*, 1998

[9] Hongtu Jiang and Viktor Owall, Controller Synthesis in Hardware Accelerator Design for Video Segmentation, *SSoCC*, 2004.

[10] K. Johnston, D. Gribbon and D. Bailey, Implementing mage Processing Algorithms on FPGAs, *Proceedings of the Eleventh Electronics New Zealand Conference*, Palmerston North, Nov. 2004.

[11] Craig Sanderson and Dave Shad, FPGAs Supplant Processors and ASICs in Advanced Imaging Applications, *FPGA and Programmable Logic Journal*, 2005.

[12] C. Stauffer and W. E. L. Grimson, Adaptive background mixture models for real-time tracking, *IEEE Conference on Computer Vision and Pattern Recognition*, 1999.

[13] C. Wren, A. Azarbayejani, T. Darrel and A. Pentland, Pfinder: Real-time tracking of human body, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997.

[14] Inc. Xilinx, Virtex II Platform Field Programmable Gate Arrays Data sheet, *http://direct.xilinx.com/bvdocs/publications*, March 2005.

[15] Pavel Zemeik, Hardware Acceleration of Graphics and Imaging Algorithms using FPGAs. *Proceedings of Spring Conference on Computer Graphics*, 2002.