A Video Compression Algorithm Suitable for Security Applications

Joseph Mirsky, Elad Gabay and Dr. Shimon Mizrachi
Electronics Engineering Department,
Jerusalem College of Technology
21 Havaad Haleumi Street, Jerusalem, Israel
yossim@geocities.com, gabayeh@gmail.com, shimon@aka.co.il

Abstract

Producing a compression unit for digital security video cameras poses a unique challenge; ideally the encoder should be small and inexpensive, yet powerful enough to compress and transmit live video. In this paper, we will present a BTC based video compression algorithm that solves this problem. The algorithm is of low computational complexity, and is suitable for easy implementation in a small FPGA, or other hardware based designs.

Keywords: BTC, image compression, video compression, lossy compression, FPGA, security cameras.

1. Introduction

The compression units attached to digital security cameras present a unique challenge in the world of digital video compression. Most compression systems' encoding processes are either slow or require a large amount of computational power. At the same time, the decoder must be inexpensive and fast. However, in the security industry, with many security cameras in use at a single site, it is the compression units that must be as inexpensive as possible. At the same time, it essential that they be fast and powerful enough to compress the video live.

The task is slightly less daunting when taking into consideration the unique aspects of video surveillance. Generally, security cameras are placed in stationary positions with little or no change in the backgrounds of the images. Furthermore, security videos run at slower frame rates and have lower quality standards then other forms of digital video, such as movies. Finally, the nature of security camera installations is such, that compressed video need only be sent to a single viewing station or computer. This computer can provide a powerful decoding platform, facilitating the use of filters to improve the image quality before displaying the video.

We have developed a BTC based video compression algorithm that produces quality digital video suitable for security applications. The algorithm achieves low computational complexity by primarily performing addition, subtraction and shift operations. The algorithm was developed with easy and simple implementation in a hardware setting in mind, and was successfully implemented and run on an FPGA.

The second section will introduce the AMBTC and the PBI_AMBTC. The third and fourth sections will present our image and video compression proposals. The fifth section will describe the implemented algorithm and the results. We will conclude in the sixth.

2. Block Truncation Coding, AMBTC, Prediction, and Homogenous Blocks

BTC, or Block Truncation Coding, is a simple lossy compression method introduced by Delp and Mitchel [1]. Every pixel in a standard grayscale image can be set at one of 256 different levels of gray, giving a data rate of 8 bits per pixel (bpp). BTC compresses the image to two bits per pixel. Lema and Mitchel [2] presented a fast form of BTC called Absolute Moment Block Truncation Coding (AMBTC) that achieves similar results to BTC with less computational complexity.

Using AMBTC, the image is divided into non-overlapping blocks, usually of 4 by 4 pixels. The average gray level in the block $\bar{x}$ is calculated using equation (1).

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

$\bar{x}$ is then used as a threshold to sort the block's pixels into one of two categories. The pixels whose values are equal to or above the average gray level are placed in the higher group. The remaining pixels whose values are lower than $\bar{x}$ are placed in the lower group. Using equations (2) and (3) the higher and lower averages of each block are calculated, where '*k*' is the number of pixels whose values are greater or equal to $\bar{x}$.

$$\bar{x}_H = \frac{1}{k}\sum_{x_i \geq \bar{x}} x_i \tag{2}$$

$$\bar{x}_L = \frac{1}{16-k}\sum_{x_i < \bar{x}} x_i \tag{3}$$

Next, a bit map is produced by quantizing the block into two levels. If a pixel is equal to or above the block's gray average value, then the bit '1' is stored in the bit map. In the remaining spots of the bit map '0' is stored.

The upper and lower averages, at 8 bits each, as well as the 16 bit bit map, are transmitted to the decoder. To reconstruct the block, the '1' bits in the bit map are replaced by the upper average and the '0' bits are replaced by the lower average. The total number of bits required to encode a block size of 4 by 4 pixels is 8+8+16=32 bits or 2 bpp.

PBI_AMBTC

Somasundaram and Raj proposed an AMBTC based compression scheme, PBI_AMBTC, that greatly improves the bit per pixel rate, while maintaining low computational complexity [3]. The algorithm makes use of three techniques; prediction, bit plane omission and interpolation.

Prediction takes advantage of the correlation of neighbouring blocks to each other. The current block is compared to the left, upper left, upper, and upper right blocks as shown in Figure 1. Using equation (4), where 'x_i' and 'y_i' are the ith pixels in the block, the Euclidian error between the current block and its neighbours is calculated. If the minimum Euclidian distance is beneath a certain threshold, then a 3 bit code is generated to inform the decoder which neighbouring block should replace the current block. In order to avoid false matches, the use of prediction necessitates the introduction of a decoder and memory into the encoder.

$$d(x,y)=\sum_{i=1}^{n}(x_i-y_i)^2 \tag{4}$$

Upper Left Block	Upper Block	Upper Right Block
Left Block	**Current Block**	

Figure 1: Position of Prediction Blocks.

If prediction does not succeed then bit plane omission is attempted. If the difference between the upper and lower block averages is within a certain threshold, the bit map and the upper and lower averages are omitted and the block's average $\bar{x}$ is sent instead. During decoding, all of the block's pixels are given the value of $\bar{x}$, creating a homogenous block. The most effective threshold for bit plane omission is 20 [4].

The final technique Somasundaram and Raj proposed is interpolation. If bit plane omission does not succeed then BTC based compression must be used. However, to further reduce the bits per pixel, half of the bits in the bit map are omitted as shown in Table 1. In the decoding stage, the omitted pixel's values are reconstructed through interpolation of the surrounding pixels as shown in equation (5). A further 4 bits are saved by excluding the 2 least significant digits of the upper and lower averages.

X_1	$\boldsymbol{x_5}$	X_9	$\boldsymbol{x_{13}}$
$\boldsymbol{x_2}$	X_6	$\boldsymbol{x_{10}}$	X_{14}
X_3	$\boldsymbol{x_7}$	X_{11}	$\boldsymbol{x_{15}}$
$\boldsymbol{x_4}$	X_8	$\boldsymbol{x_{12}}$	X_{16}

Figure 2: Omitted bits in italics.

i	**Interpolation Equation**
2	$x_i=\frac{1}{3}(x_{i-1}+x_{i+1}+x_{i+4})$
4	$x_i=\frac{1}{2}(x_{i-1}+x_{i+4})$
5	$x_i=\frac{1}{3}(x_{i-4}+x_{i+1}+x_{i+4})$
7,10	$x_i=\frac{1}{4}(x_{i-4}+x_{i-1}+x_{i+1}+x_{i+4})$
12	$x_i=\frac{1}{3}(x_{i-4}+x_{i-1}+x_{i+4})$
13	$x_i=\frac{1}{2}(x_{i-4}+x_{i+1})$
15	$x_i=\frac{1}{3}(x_{i-1}+x_{i+1}+x_{i-4})$

Table 1: Interpolation Equations.

As seen, using PBI_AMBTC, the bit per pixel rate can be reduced considerably. Instead of 2 bpp, a block compressed using prediction requires 0.1875 bpp. If predication does not succeed then bit plane omission at 0.5 bpp is used. If all else fails, interpolation still improves the bit per pixel rate to 1.375.

3. Image Compression

To compress images we make use of PBI_AMBTC with some improvements and modifications. The most significant change is the use of common bit maps; as such, our image compression algorithm is named CBM_AMBTC.

To lessen the computational complexity of the prediction technique, instead of calculating the Euclidian distance, we propose summing the difference between two neighbouring blocks as shown in equation (5). This will avoid the need of performing squares, a complex operation, and entail only addition and subtraction instead. Using this method and a threshold of 40, the image quality is also improved slightly by an average of 0.02 dB.

$$d(x,y)=\sum_{i=1}^{n}\|x_i-y_i\| \tag{5}$$

Instead of sending an upper and lower average, it is more efficient to transmit the data in the form of sigma and the block's average gray level $\bar{x}$. The higher and lower averages are reconstructed in the decoder using equations (6) and (7).

$$\bar{x}_H=\sigma+\bar{x} \tag{6}$$

$$\bar{x}_L=\sigma-\bar{x} \tag{7}$$

The block's sigma and gray average are calculated with low computational complexity as follows: The block's higher and lower averages are calculated as normal for AMBTC. The difference and sum of the two averages is determined and their values are then divided by two. Implementing in logic a divide by two operation is simply a one bit shift to the right. Sigma and $\bar{x}$ are calculated in equations (8) and (9).

$$\sigma=\frac{\bar{x}_H-\bar{x}_L}{2} \tag{8}$$

$$\bar{x}=\frac{\bar{x}_H+\bar{x}_L}{2} \tag{9}$$

As the difference between the higher and lower averages of a block rarely exceed 128, we can limit the value of sigma to a maximum of 63. By removing the 2 least significant bits of σ and $\bar{x}$, we can transmit σ in 4 bits and $\bar{x}$ in 6, saving an additional 2 bits. When using bit plane omission, the least significant bit of the mean block value is dropped as well.

Removing the 2 least significant bits from σ and $\bar{x}$ (or the higher and lower averages) will often result in lower than accurate values appearing in the decoder. To compensate, a set value should be added to the σ and $\bar{x}$ in the decoding stage. For best results, 1 should be added to $\bar{x}$, if σ has attained its maximum value of 60 (111100_2) it should be increased by 19, otherwise by 7.

Common Bit Maps

To save bits, Somasundaram and Raj proposed dropping half of the bit map's bits, and interpolating their values in the decoder. This method decreases the quality of the images by over 1 dB. We propose a different method that decreases the size of the bit map to a similar extent without so drastically reducing the image quality.

It is extraneous to use 16 bits to send the bit map, as certain configurations are unlikely to occur. For example, a bit map of alternating high and low averages as seen in Figure 3, is unlikely to occur.

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

Figure 3: Unlikely bit map configuration.

With a bit map size of 16 bits, there are 65,536 possible combinations; many of them rarely, if ever, transpire. One can create a set of the most common bit map configurations and use a short code to represent each bit map. In cases where a block's bit map does not have an exact match in the set, then the closest bit map is found using equation (10); where 'x_i' and 'y_i' are the i^{th} bits in the bit map. The common bit map which shares the most '1' bits and '0' bits with the current block's bit map, and therefore has the lowest 'Error', is the match.

$$Error = \sum_{i=1}^{16} \|x_i - y_i\| \qquad (10)$$

Displayed in Figure 4 is a group of 12 pictures, typical of those captured by security cameras, that we analyzed for common bit maps. Each grayscale image is 320 x 240 pixels and can be divided up into 4,800 blocks of 4 by 4 pixels, each block with its own corresponding bit map. This created a total of 57,600 bit maps. The most common bit maps are set aside and each bit map is represented by a binary code word. The results are summarized in Table 2. By using a set of 256 common bit maps, nearly 79% of the images' bit maps have exact matches, with the remaining bit maps being assigned the closest match from the set. This causes an average reduction in image quality of 0.4 dB compared to the 1 dB of interpolation for same the bit per pixel rate.

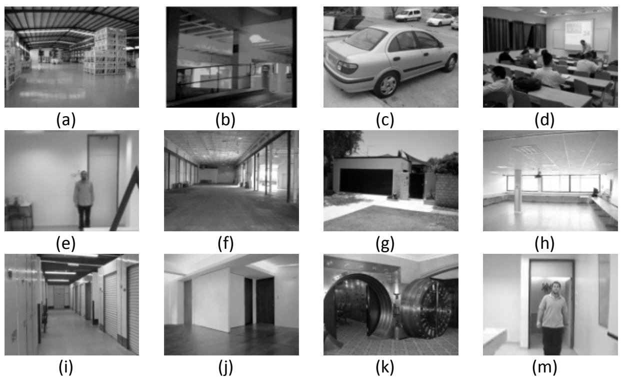


Figure 4: Typical security camera images.

Size of set of common bit maps	Code Size per bit map	Percentage of exact matches
128	7	69.5%
256	8	78.6%
512	9	85.9%

Table 2: Results of Common bit maps.

Implementing the common bit map technique on an FPGA or similar digital logic devices is simple and effective. One can calculate in advance the closest common bit map match for every bit map configuration. Every one of the 65,536 possibilities has an appropriate common bit map code associated with it. With a set of 256 common bit maps, the technique can be implemented in the form of a 16 bit address, 8 bit output, ROM. The current bit map's configuration selects the correct address, and the ROM's output is the code for the common bit map that is the closest match.

Deblocking Filter

A common consequence of using a block based compression method is the formation of noticeable distortions in the transition zones between one block and another. To smooth these distortions, as the last step in the image decoding process we added a deblocking filter [5]. The filter detects drastic changes between pixel values on the edges of one block to another and smoothes them over using a Gaussian filter. While this does not necessarily improve the quality of an image in terms of PSNR, the subjective, "pleasing to the eye" appearance of the image is greatly enhanced.

4. Video Compression

We developed a video compression scheme based on the CBM_AMBTC, and consequently will refer to it as VCBM_AMBTC ('V'–Video).

The simplest form of video compression is to take advantage of the image redundancy in a temporal sense. The first frame is set as a reference frame and entirely encoded using the image compression algorithm. Each subsequent frame is compared to the frame that preceded it. Using equation (5), each block is compared to the block in the same location of the previous frame. If the difference is within a threshold, the two blocks are similar and a flag is sent to the decoder; otherwise the block must be encoded using CBM_AMBTC. Comparing the two blocks before encoding them using the CBM_AMBTC prevents quantization noise from entering the images, and prevents the encoder from performing unnecessary compression operations on matching blocks.

To fix transmission errors and prevent the build-up of quantization errors, the frame comparisons are not allowed to continue indefinitely. Rather, the frames are compressed in groups, or packets. The first frame of the packet is a reference frame and compressed using CBM_AMBTC. The subsequent frames are compared to those previous, and so on, until the end of the packet. The cycle then starts over again with a new packet and new reference frame.

As most security cameras are stationary and there are long periods with little or no movement, there will often be many blocks, or even entire frames, in a packet that do not change. Rather than transmit a flag every time a block remains unchanged, one can store the number of matches in a counter. Every time a change between blocks is detected, first the contents of the match counter is transmitted, then the block is compressed.

A common video packet size is 32 frames. The maximum amount of block matches in such a packet is 31 x 4,800 = 148,800, which requires a counter size of 20 bits. To save bits, instead of always transmitting a counter of 20 bits, one can enable the decoder to work with variable counter sizes through the use of flags. Efficient variable counter sizes for a packet of 32 frames is described in Table 3.

Flag	Counter Size (bits)	Block Matches
'00'	2	4
'01'	5	32
'10'	9	512
'11'	20	1,048,576

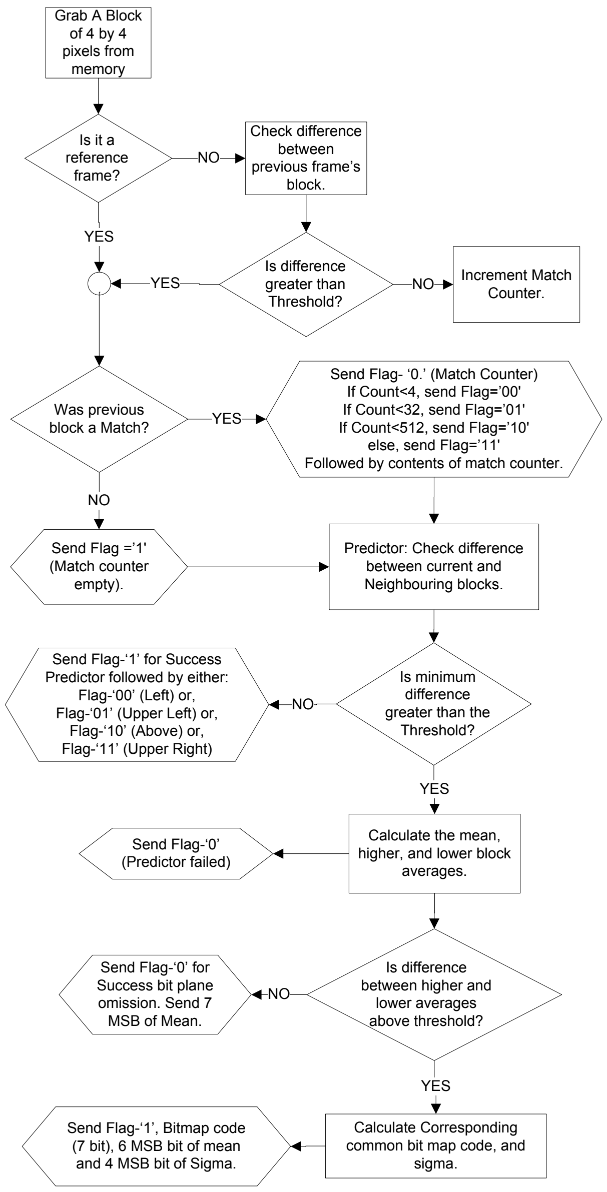
Table 3: Variable counter sizes and flags.

5. Algorithm Implementation and Results

We tested our algorithm using software based platforms such as MATLAB. We found that CBM_AMBTC produced better compression rates while maintaining higher image quality when compared to other low computational complexity compression methods in existence. For example, on images (b), (c), (e), and (m), CBM_AMBTC produced an average bit per pixel rate of 0.527 with a PSNR of 36.32 dB. Even for 'classic' images such as Lena, Jet, or Peppers, which are not characteristic of security type images, CBM_AMBTC performed better than other comparable compression methods. The results are summarized in Table 4 and partially based on the results reported by Somasundaram and Raj for a predictor threshold of 200 [3].

In MATLAB simulations, VCBM_AMBTC produced good videos with good image quality and a compression ratio of 235, or 0.0339 bpp.

We translated our algorithm to VHDL and implemented it on an FPGA. The implemented VCBM_AMBTC algorithm for a single block is presented in Figure 5.



Grab A Block of 4 by 4 pixels from memory
Is it a reference frame?
NO
Check difference between previous frame's block.
YES
YES
Is difference greater than Threshold?
NO
Increment Match Counter.
Was previous block a Match?
YES
Send Flag- '0.' (Match Counter) If Count<4, send Flag='00' If Count<32, send Flag='01' If Count<512, send Flag='10' else, send Flag='11' Followed by contents of match counter.
NO
Send Flag ='1' (Match counter empty).
Predictor: Check difference between current and Neighbouring blocks.
Send Flag-'1' for Success Predictor followed by either: Flag-'00' (Left) or, Flag-'01' (Upper Left) or, Flag-'10' (Above) or, Flag-'11' (Upper Right)
NO
Is minimum difference greater than the Threshold?
YES
Send Flag-'0' (Predictor failed)
Calculate the mean, higher, and lower block averages.
Send Flag-'0' for Success bit plane omission. Send 7 MSB of Mean.
NO
Is difference between higher and lower averages above threshold?
YES
Send Flag-'1', Bitmap code (7 bit), 6 MSB bit of mean and 4 MSB bit of Sigma.
Calculate Corresponding common bit map code, and sigma.

Figure 5: VCBM_AMBTC algorithm.

We used Xilinx's SP 305 Developers board which includes Spartan 3 XC3S1500-FG676, Cypress CY7C1354B 9 Mbit external SRAM, and a RS-232 UART serial port. The Spartan 3 was our FPGA, the SRAM was used to store the previous, current, and decoded frames, and the RS-232 UART was used to transmit the compressed data to the computer workstation where it was decoded. The FPGA uses a clock of 100 MHz and captured video from Omnivision's M3188A black and white digital camera at a rate of 10 frames per second. Post place and route results of Xilinx's ISE 10.1 showed that our design can fit into a Spartan 3 as small as XC3S400-4TQ144.

The threshold values that we used were as follows: Predictor – 40, bit plane omission – 20, block match between frames – 160. The image size was 320 by 240 pixels with a packet size of 32 frames. The compressed video was transmitted live by the FPGA's UART port with a BAUD rate of 115,200.

The digital camera was placed in a stationary position facing a room with a static background. A screenshot typical of the output of the camera and compression system can be seen in Figure 6. We remind the reader that it is difficult to judge video quality from a still image.

We tested our system's compression capabilities by reproducing situations typical of security camera use. Little or no movement or change in the video was simulated by a solitary person walking slowly across the room with a gap of 5-7 seconds between each person. The average compression ratio achieved was 377 with a bpp of 0.0212. Events consisting of large and continuous movements or changes were simulated by several people passing near the camera, with little or no gaps in between them. The average compression ratio obtained was 173 with a bpp of 0.0462.

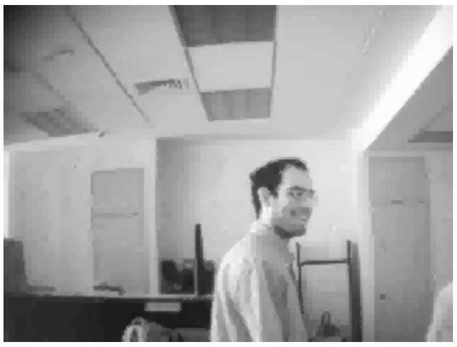


Figure 6: Typical Screenshot.

6. Conclusion

A low bit rate video compression scheme has been presented that combines prediction, bit plane omission, common bit maps, sigma and mean, comparisons between frames, run length encoding, and a deblocking filter. The implemented VCBM_AMBTC algorithm uses low computational complexity and produces digital video of good quality suitable for applications such as security cameras. Due to the scheme's high compression ratio, the most basic and inexpensive of transmission methods can be used to stream the video live. At the same time, VCBM_AMBTC can be implemented on small FPGA's and other low-cost digital logic devices.

Method	**AMBTC**		**PBI_AMBTC**		**YCH** [6]		**CBM_AMBTC**	
Images	BPP	PSNR (dB)	BPP	PSNR (dB)	BPP	PSNR (dB)	BPP	PSNR
Jet	2.0	31.42	0.54	30.53	0.51	27.69	0.52	31.17
Lena	2.0	33.25	0.61	31.00	0.57	28.47	0.56	31.77
Peppers	2.0	33.44	0.62	30.54	0.61	28.54	0.59	31.69
Average	2.0	32.70	0.59	30.69	0.56	28.23	0.56	31.54

Table 4: Comparison of compression algorithms.

References

1. **E.J. Delp, O.R. Mitchell.** *Image Compression Using Block Truncation Coding.* s.l. : IEEE trans. Communications, 1979. Vol. 27.

2. **M. D. Lena, and O. R. Mitchell.** *Absolute Moment Block Truncation Coding and its application to color Images.* s.l. : IEE Trans. On communications, 1984. Vol. 32.

3. **Somasundaram, K.and I. Kaspar Raj.** *An Image compression Scheme based on Predictive and Interpolative Absolute.* Tamilnadu, India : GVIP Journal, December 2006. Vol. 6.

4. **Somasundaram, K.and I. Kaspar Raj.** *Low Computational Image Compression Scheme based on Absolute Moment Block Truncation Coding.* May 2006. Vol. 13.

5. **Richardson, Iain E.G.** *H.264 and MPEG-4 Video Compression. Video Coding for Next Generation Multimedia.* 2005.

6. **Hu, Yu-Chen.** *Low complexity and low bit-rate image compression scheme based on Absolute Moment Block Truncation Coding.* s.l. : Optical Engineering, 2003. Vols. SPOIE Vol. 42 No. 7, pages 1964-1975.