Selection of bi-level image compression method for reduction of communication energy in wireless visual sensor networks

Khursheed Khursheed, Muhammad Imran, Naeem Ahmad, Mattias O'Nils

Department of Information Technology and Media, Division of Electronics Design, Mid Sweden University Sundsvall, Sweden

{khursheed.khursheed,naeem.ahmad,muhammad.imran, mattias.onils}@miun.se

ABSTRACT

Wireless Visual Sensor Network (WVSN) is an emerging field which combines image sensor, on board computation unit, communication component and energy source. Compared to the traditional wireless sensor network, which operates on one dimensional data, such as temperature, pressure values etc., WVSN operates on two dimensional data (images) which requires higher processing power and communication bandwidth. Normally, WVSNs are deployed in areas where installation of wired solutions is not feasible. The energy budget in these networks is limited to the batteries, because of the wireless nature of the application. Due to the limited availability of energy, the processing at Visual Sensor Nodes (VSN) and communication from VSN to server should consume as low energy as possible. Transmission of raw images wirelessly consumes a lot of energy and requires higher communication bandwidth. Data compression methods reduce data efficiently and hence will be effective in reducing communication cost in WVSN. In this paper, we have compared the compression efficiency and complexity of six well known bi-level image compression methods. The focus is to determine the compression algorithms which can efficiently compress bi-level images and their computational complexity is suitable for computational platform used in WVSNs. These results can be used as a road map for selection of compression methods for different sets of constraints in WVSN.

Keywords: Wireless Visual Sensor Network, Communication Energy Consumption, Computational Complexity, Visual Sensor Nodes, Image Compression.

1. INTRODUCTION

Wireless Visual Sensor Network (WVSN) is formed by deploying many Visual Sensor Nodes (VSNs) in the field. Typically VSN in WVSN consists of an image sensor for acquiring images of the area of interest, a processor for local image processing and a transceiver for communicating the results to the central base station. WVSNs are most suitable for those applications which have a limited availability of power and in those cases where it is inconvenient to modify the location of the VSN or to frequently change the batteries. Based on the technological developments in image sensors, sensor networking, distributed processing, low power processing and embedded systems, a VSN can perform its tasks using a suitable Field Programmable Gate Array (FPGA)/Microcontroller, a wireless link and with limited storage. In order to be operational for a reasonable lifetime, low power sensing, processing and communication hardware is required for the VSN. The VSN must perform complex tasks (e.g. image compression, labeling and object features extraction) by using an embedded processing unit and must transmit the results wirelessly, but, unfortunately, it possess very limited energy resources. The energy consumption and bandwidth are major constraints in WVSN.

Both on board processing and communication influence the total energy consumption of the VSN. Transmitting the results from VSN, without local processing, reduces the processing energy but, it results in higher communication energy due to the transmission of large chunks of raw data. On the other hand, if all the processing is performed locally at the VSN and only the final results are transmitted, then the communication energy is reduced tremendously, however, this causes an increase in the processing energy because of increased processing at the VSN.

A previous study on intelligence partitioning between the VSN and the server[1] concludes that the selection of a suitable intelligence partitioning strategy reduces the total energy consumption of the VSN. In addition, the results in[3] show that coding the binary image after preprocessing and segmentation is a good alternative in order to achieve a general architecture for a WVSN. Based on this general architecture, the focus in this paper is on selecting the most energy efficient bi-level image coding method for further reducing the communication energy consumption.

In a machine vision scenario, images contain a few objects located at random locations. Our focus in this work is on the compression analysis of these kinds of images and in comparing different bi-level image compression methods, based on various features of these images. The size, shape and locations of the objects in the images could vary significantly. Thus, in order to analyse image compression methods based on these kinds of images, a very rich set of statistically generated images is a mandatory requirement. The statistically generated images must have objects consisting of a variety of features such as size, location, shape and number of objects in the images. For this analysis, images have been randomly generated which include all these features, and these are explained in detail in the results section.

Based on these statistically generated images, the requirement is to determine the behavior of different compression methods from a variety of aspects. Specifically, the desire is to analyze the effect of an increase in the number of objects, size of objects and shape of objects, on the compression efficiency and communication energy for the different compression algorithms. Our aim is to discover a compression method that is resilient to these different features in relation to the objects in the statistically generated images. Such a compression method will be suitable for energy constrained embedded systems.

The remainder of the paper is organized as follows. In section 2, related work is provided. In section 3, a very brief introduction to the computational complexity of all the considered compression methods is given. In section 4, the results are presented based on the compression efficiency of all the compression methods. This is followed by a discussion of the execution times and implementation details of all the considered compression methods in section 5. Finally Section 6 concludes the paper.

2. RELATED Work

Although the work on image compression and communication over wireless networks is both widespread and venerable, but the recent emergence of WVSN for large scale surveillance applications has imposed new challenges because of the stringent limitations on memory, processing speed, communication bandwidth and energy consumption in VSN. There are many bi-level image coding methods, however, presently there is no analysis of which of them presents the best total energy characteristics for embedded wireless use. For instance, the latest and most efficient standard for compressing bi-level images from Joint Bi-level Image Experts Group (JBIG) is JBIG2[11]. However, the focus in the design of JBIG2 has been on the efficient storage of different kind of images, which has been fulfilled. The problem associated with JBIG2 for embedded use is in relation to its higher computational complexity, which thus causes a higher processing energy.

Few researchers have compared the image compression methods, but the problem with all of these is that the focus is only in relation to an investigation of compression efficiency. One such comparison of international standards for lossless still image compression has been conducted in[23]. They have thoroughly investigated the compression efficiency of all the well-known compression methods available at that time. One problem associated with[23]is the absence of the latest standard JBIG2 in their comparison. Another issue is the lack of consideration of the execution time of the compression methods. Energy consumption of the embedded implementation of an algorithm depends on its execution time, which has not been investigated in[23.]

Even in[24] where both the compression efficiency and execution time are investigated, the focus has not been on the energy consumption of the embedded implementation of the compression methods. In addition, they have investigated the efficiency of compression methods based on textual data, which is different in relation to the images in a machine vision scenario.

Another study on the comparison of compression methods has been conducted in[25]. They have applied many compression standards and some compression algorithms on different medical images. They have compared both the compression efficiency and execution time of the compression techniques. However, it is still the case that no consideration has been given to the energy consumption of the embedded implementation. They have pointed out that the compression performance depends on the type of the images which means that these results cannot be directly applied to machine vision applications because of the different types of images. In machine vision applications, the images are quite different from scanned textual and medical images and a thorough study for these kinds of images is required because the efficiency of bi-level image compression standards varies from image to image.

This paper investigates different energy efficient bi-level image coding methods for machine vision embedded applications. We have studied six well known bi-level image compression methods and analyzed the communication energy consumption (based on compression efficiency) and computational complexity for each method. The considered methods are JBIG2[11], Gzip, Gzip_pack (for packed images), CCITT Group 3, CCITT Group 4[22] and one of the rectangular compression methods[9]. The scope of this paper is limited to the lossless compression schemes.

3. BI-LEVEL COMPRESSION METHODS

A number of lossless data (image) compression algorithms exist, which can compress almost any kind of data. The well-known algorithms are rectangular edge coding[4], rectangular coding (REC)[5], Modified Relative Element Address Designate (READ) coding[6], Ziv-Lempel algorithms [8], Efficient Partitioning into Rectangular Regions[9], Arithmetic coding[10] etc. These algorithms are lossless in the sense that the compressed images retain all the information of the original data and an exact replica of the images can be reproduced at the receiver side. JBIG2[11] is a lossless image compression standard from JBIG. It is widely implemented in fax machines. The JBIG2[11] compression standard is based on a form of Arithmetic coding[13] called MQ coder and can be both lossy and lossless. The MQ coder used in JBIG2 is an adaptive binary arithmetic coder, which is characterized by a multiplication free approximation and a renormalization-driven update of the probability estimator, and bit stuffing introduced by a Q-coder[12].

The Gzip (GNU zip) is a lossless universal standard used for document compression (could be images) and is based on Ziv-Lempel algorithms[8]. We have used Gzip in two ways, namely the Gzip and the Gzip_pack. It is the same implementation, however, for the Gzip standard black and white images (one byte per pixel) have been used as the input whereas for the Gzip_pack, the first 8 pixels of the image have been packed into one byte after which it has been compressed using Gzip command. The implementations for both the Gzip and Gzip_pack are exactly similar (standard Gzip command in Linux) the only difference being in the input image data format.

The International Telegraph and Telephone Consultative Committee (CCITT) is a standard organization that has developed a series of communications protocols for the facsimile transmission of black-and-white images over telephone lines and data networks. These protocols are officially known as the CCITT T.4 and T.6 standards[22] and are more commonly referred to as CCITT Group3 and Group4 (also known as Fax3 and Fax4) compressions, respectively.

3.1 CCITT Group 3 (2D) and Group 4

CCITT Group 3 2-Dimensional (2D) and Group 4 are based on the Huffman coding and are almost the same, with only a few differences. Group 3 2D compresses k-1 (typical value for k is 2 or 4) lines in a 2D manner and the kth line is coded in a 1-Dimensional (1D) manner. On the other hand, in Group 4 the image is always compressed in a 2D manner. Another difference is that the Group 3 2D inserts the End Of Line (EOL) code after the coding of each line of the image so as to prevent propagation errors. All other aspects are almost the same. The complexity of these two methods depends on the following operations.

- Comparison for detecting Changing Picture Element (CPE)
- Calculating black and white runs based on CPE
- Calculating addresses for Huffman codes in memory (Involves shift right and subtraction operations)
- Accessing Huffman codes from memory and placing the codes in the output buffer
- Writing the header and the output buffer into the TIFF file

3.2 Efficient Partitioning into Rectangular Regions

The author in[9] proposed a bi-level image compression method which does not involve any kind of codes (e.g. Huffman codes) or any other probability estimation and statistical operations. They partitioned black regions of the image into non-overlapping rectangular regions and encoded the locations of two vertices of each rectangle. For encoding the location of the vertices, it is necessarily needed to find the number of bits required for its binary representation. In addition, for each pixel of the input image, there is a comparison operation for determining, whether or not this pixel forms part of a non-overlapping rectangle. After coding the vertices of one non-overlapping rectangle, the search for finding the vertices of another non-overlapping rectangle is started. In this way the whole images is coded. The computational complexity of the Rectangular compression method is affected by the following operations.

- Comparisons for determining vertices of non-overlapping rectangles.
- Calculating the exact number of bits required for storing the locations of the vertices of non-overlapping rectangles.
- Finding the binary value of the vertices and placing these values into the output buffer.
- At the end of each line a binary 1 is inserted in the output buffer.

3.3 The Gzip

The Gzip universal compression standard is based on LZ77[8], which is a dictionary based lossless compression scheme and is usually used for text compression. The Gzip uses a sliding window of size N where N could be of any size. In the implementation of[18], N is 32 Kilo Bytes (KB). When the next sequence of characters which are to be compressed is identical to the one that can be found within the sliding window, the sequence of characters is replaced by two numbers: a distance, representing how far back into the window the sequence starts, and a length, representing the number of characters for which the sequence is identical. Thus long sequences are replaced by only two binary numbers of a few bytes. In this way, the compression is achieved. The following operations involved in the Gzip compression are the source for its computational complexity.

- Comparisons for detecting whether the current byte is present in the sliding window
- Detecting how many bytes next to the current byte in the input file are identical to the one present in the sliding window
- Storing the address of an identical byte in the sliding window.
- Saving the information about how many bytes are identical from the current byte in the input file and the sliding window.

3.4 The JBIG2

The compression algorithm employed in JBIG2 is arithmetic coding[10]. The most important advantages of arithmetic coding are its flexibility and optimality. It is flexible in the sense that it can be used in conjunction with any probability model, which can provide a sequence of event probabilities. The main disadvantage of arithmetic coding is its high encoding time. In order to achieve full precision, an arithmetic operation requires one multiplication for every event in the input file. The focus of some research has been on the approximation of the multiplication operation in arithmetic coding[19-20]. Due to these alternate solutions for this mathematical operation and the present availability of fast hardware, arithmetic operations no longer limit the performance of arithmetic coding. Thus, in order to analyze the computational complexity of arithmetic coding, it is necessary to consider all the other operations involved in arithmetic coding.

Amir Said[14] extensively analyzed the computational complexity of arithmetic coding and identified the following sources of computational complexity in relation to arithmetic coding.

- Interval update and arithmetic operations.
- Symbol decoding
- Interval renormalization
- Carry propagation and bit moves
- Probability estimation
- Support for non-binary symbol alphabets

4. EVALUATION OF CODING PERFORMANCE

The aim of all compression schemes is to transform an image into a compressed form. The main objective is to preserve the information content and to reduce the data as much as possible. The principal parameter of a compression technique is the compression efficiency. It is simple to design a compression algorithm having a low bit rate, but the real challenge is to also preserve the quality of the reconstructed image. Thus, the two main criterions in relation to measuring the performance of an image compression algorithm are the compression efficiency and distortion, caused by the compression algorithm. As the scope of this paper is limited to the analysis of bi-level lossless compression techniques only, the distortion part of performance will not be discussed. Another performance measurement criterion is the speed of the compression and decompression process, which is covered in section 5.

The compression efficiency of bi-level image compression algorithms is highly dependent on the transitions from the black to white pixels and from the white to black pixels in the input images. In order to determine the performance of the compression algorithms under observation in this paper, random images of size 640x400 have been created with random features of the objects in a black background. In one set of randomly generated images, the number of objects in the images has been increased. Similarly, in another set of images, the number of objects has remained fixed but there has been an increase in their sizes. Both of these sets of images contain objects of various shapes such as circles, semi-circles, quarters of circles, ellipses, semi-ellipses, quarters of ellipses, rectangles and curves. Our goal is to determine the compression behavior of the compression methods, for various kinds of changes (fully random) in the input images such as the number, shape and the size of the objects.

4.1 Increasing number of objects in the images

For each shape of a white object, such as the circles, ellipses, rectangles etc, 10 images have been randomly generated by using Matlab script and there has been a 10% increase in the number of objects in each successive image. Five sample images with one shape i.e. a quarter of an ellipse are shown in Figure 1. The randomness in the placement and the increase in the number of objects in the images is obvious in Figure 1.

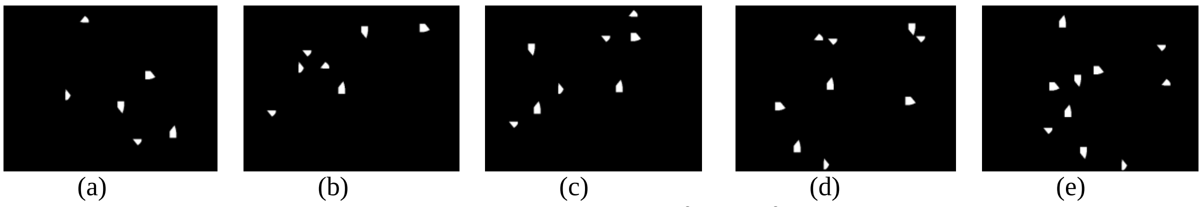

Figure 1: Images with random placement and increased number of objects from (a) to (e).

The compression efficiency of the considered algorithms has been analyzed for eight different object shapes in the images. For each shape, 10 images have been compressed with a 10 % increase in the number of objects in each successive image. The trend of compressing 10 randomly generated images, for a quarter of an ellipse, for all six compression algorithms is shown in Figure 2. The vertical-axis in Figure 2 shows the compressed file size in Bytes while the horizontal-axis shows the number of objects in the image. The number of objects is increasing from left to right and from 1 to 10.

It is obvious in Figure 2 that the compressed file size is increasing by increasing the number of objects in the images, for all the compression methods. In addition, the size of the compressed file from the Rectangular compression method is lower than that of the JBIG2 for an image with one object and is higher than that of the JBIG2 for all other images having more than one object. Similarly, the size of the compressed file from the Rectangular method is lower than that of the CCITT Group 4 for an image with 1 to 4 objects and is higher for an image having 5 or more objects. These trends show that some methods are more sensitive to the increase in the number of objects than is the case for others. Thus, in order to analyze the effect of increasing the number of objects for the eight different shapes, a sensitivity measure must be defined which can be used for both a quick and accurate analysis.

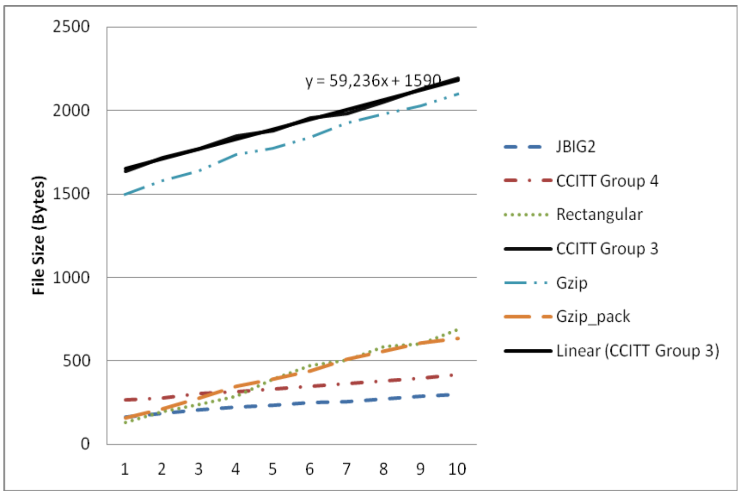

Figure 2: Effect of increasing number of objects on compression efficiency of six methods.

The topmost curve (CCITT Group 3) has been approximated by a line and, in addition, an equation has been shown for that line in Figure 2. The slope of the line shows the sensitivity of this compression method to the increase in the number of objects in the images. A higher slope value means a higher sensitivity to the increase in the number of objects. The slope of the lines has been analyzed for all the algorithms and for all the different shaped objects in the images. Instead of showing graphs by means of lines and their equations for all the object shapes, this information has been displayed in a compact form in Table 1.

The mean and standard deviation in relation to both the compressed file size and the sensitivity (slope) for all the considered compression methods for compressing 10 randomly generated images with a 10% increase in the number of objects in successive images is shown in Table 1. The mean sensitivity is calculated by taking the average of the slope of the curves for all the images with different shapes and for the increase in the number of objects. Similarly, the mean compressed file size is determined by calculating the average of the compressed file sizes for all the images with different shapes and with an increasing number of objects. The standard deviation is a self-explanatory term and is calculated for both the mean compressed file size and sensitivity.

Table 1: Mean and standard deviation of compressed file size and sensitivity for all methods

Compression Methods	Mean sensitivity (k)	Standard Deviation (σ_k)	Mean Compressed File Size (d)	Standard Deviation (σ_d)
Gzip	122	42	2092	224
Gzip_pack	86	35	576	196
Group 4	31	14	421	74
Group 3	109	42	2193	234
JBIG2	21	9	287	53
Rectangular	98	47	628	266

It must be noted in Table 1 that the mean sensitivity for both the JBIG2 and CCITT Group 4 is lower than for all the other compression methods. The standard deviation in sensitivity for both the JBIG2 and CCITT Group 4 is also lower than for all the other compression methods. It must also be observed, in the same table, that the mean compressed file size and its standard deviation for both the JBIG2 and CCITT Group 4 are lower than for all the other compression methods. Based on the observation in Table 1, the conclusion is that both the JBIG2 and CCITT Group 4 are comparatively less sensitive to the increase in number of objects of different shapes in the images.

4.2 Increasing the size of the objects in the images

For each shape of the white object, such as curves, semi-circles, ellipses, rectangles etc, 10 images have been generated by using Matlab script and the size of the objects has been increased by 10%, in each successive image. Five sample images with one shape i.e. full circle are shown in Figure 3. The randomness in the placement of the objects and the gradual increase in the size of the objects is obvious in the figure.

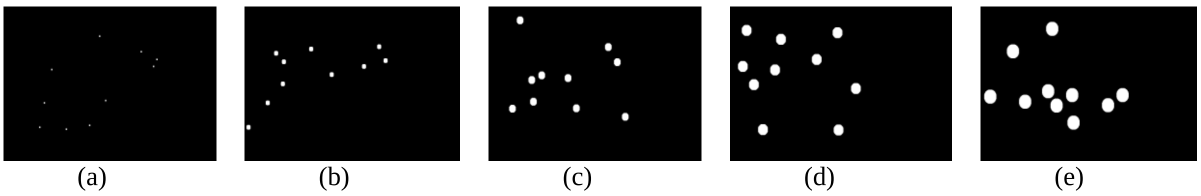

Figure 3: Images with random placement and increased size of objects from (a) to (e).

Table 2: Mean and standard deviation in file size and sensitivity for all methods

Compression Methods	Mean sensitivity (k)	Standard Deviation (σ_k)	Mean Compressed File Size (d in Bytes)	Standard Deviation (σ_d in Bytes)
Gzip	115	61	2152	396
Gzip_pack	86	57	619	356
Group 4	33	32	488	256
Group 3	110	46	2198	273
JBIG2	23	18	315	103
Rectangular	104	85	777	690

The mean and the standard deviation of the compressed file size and the sensitivity of compressing the 10 generated images with a 10% increase in the sizes of objects in successive images, for all shapes, using all the six compression algorithms under study is shown in Table 2. It must be noted in Table 2 that the mean sensitivity and its associated standard deviation for the both JBIG2 and CCITT Group 4 are lower than for all the other compression methods. It must also be observed, in the same table, that the mean compressed file size and its standard deviation for both the JBIG2 and CCITT Group 4 are also lower than for all the other compression methods. Based on the observation in Table 2, it can be concluded that both the JBIG2 and CCITT Group 4 are comparatively less sensitive to the increase in the size of objects of different shapes in images.

5. EVALUATION OF THE IMPLEMENTATION

The Rectangular compression algorithm explained in[9] is implemented using C language and is compiled and executed on the Ubuntu Operating System (OS) (Linux) running in VMware in a computer machine having Intel ® Core™2 Duo CPU with 1.86 GHz and 3 Giga Bytes (GB) Random Access Memory (RAM). For CCITT Group 3 and Group 4, the Libtiff library has been used[15].

Gzip compression is performed using the gzip command of the Ubuntu OS. JBIG2 is downloaded and built using the instructions on[16] and is executed in the Ubuntu OS. Following this, all these compression methods are used to compress many bi-level images which contain different features of the objects in the images. Each image is compressed using six compression methods and the time command of Ubuntu OS is used for determining the execution time. In order to achieve a high degree of accuracy, each image has been compressed 600 times and then the total execution time has been divided by 600 in order to determine the average value. The results are shown in the second column of Table 3.

The mean compressed file size in Table 3 is the average of the compressed file sizes of the images which have objects containing different features such as increasing size, increasing number and different shapes. Similarly the execution time is also the average value. It must be noted in Table 3 that the standard deviations for the Rectangular, Gzip, Gzip_pack and CCITT Group 3 are larger compared to the three other methods. A larger standard deviation means that these three methods are highly sensitive to different features such as the size, shape and number of objects in the input images. A high sensitivity in compressed file size of compression methods means that there is a high degree of uncertainty in the communication energy, which is an undesirable characteristic. On the other hand, the JBIG2 and CCITT Group 4 have the lowest standard deviations, which show their resilience to the variety of object features.

Table 3: Data and execution time characteristics of the compression methods

Method	T_Ubunto (ms)	Mean Compressed File Size (Bytes)	Standard Deviation in Data Size (σd Bytes)
Gzip	0.8	2123	311
Gzip_pack	0.3	598	276
Group 4	1.2	455	166
Group 3	2.0	2196	254
JBIG2	3.1	302	79
Rectangular	5.0	703	478

Figure 4 shows the complexity vs. the mean file size for the analyzed compression methods. The small circles in Figure 4 show the mean compressed file size for each method on the horizontal axis and the complexity (average execution time in ms on Ubuntu OS) on the vertical axis. The lines in Figure 4 show the standard deviation in the compressed file size for each method. It is clear from Figure 4 that the mean file sizes from Gzip and Group 3 are high compared to all the other methods. It is also evident from the same figure that the complexity is high for the Rectangular, CCITT Group 3 as well as for the JBIG2 in comparison to the other three methods. It can be stated that the Gzip_pack and Group 4 are possibly good candidates for use in energy constrained embedded systems. However, the communication energy consumption characteristics of all the methods must also be analyzed, which can then provide a more solid conclusion.

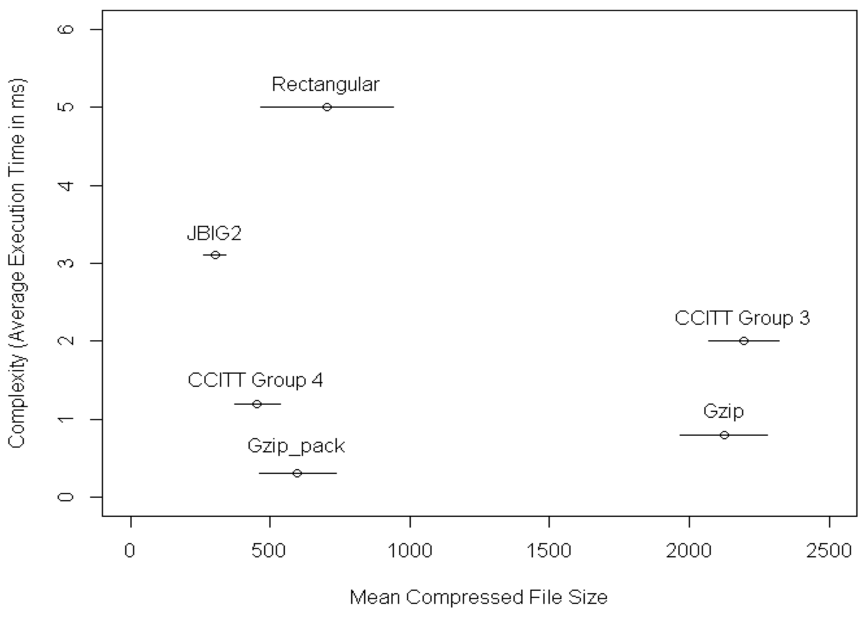

Figure 4: Complexity vs. mean compressed file size.

The communication energy consumption is dependent on the number of bytes that are required to be transmitted. The IEEE802.15.4 is considered for communicating the compressed images from VSN to the server, which has been used in the SENTIO32[17] for transmitting the results. T_IEEE in Table 4 represents the transmission time of IEEE802.15.4 and is calculated using Equation (1). T_IEEE in Table 4 is undeniably dependent on the number of bytes (Mean compressed file size from Table 3) that must be transmitted. The standard deviation in T_IEEE is calculated by using the standard deviation in the data size from Table 3. E_IEEE and the standard deviations in the E_IEEE in Table 4 are the energy consumption characteristics of the IEEE802.15.4 based on transmitting a specific amount of data. The power consumption of radio transceiver is 72 mW which is calculated using the current and voltage characteristics from its data sheet.

$$\text{T\_IEEE802.15.4} = (X+19)*0.000032 + 0.000192 \qquad (1)$$

In Equation (1) X is the number of bytes transmitted and 19 is the overhead involved due to the inclusion of header information in each packet. Factor 0.000032 in Equation (1) is the processing time of one packet while 0.000192 is the settling time of the transceiver.

Table 4: Transmission time and communication energy of the compression methods

Method	T_IEEE (ms)	E_IEEE (mJ)	Standard Deviation in T_IEEE (ms)	Standard Deviation In E_IEEE (mJ)
Gzip	82	5.9	14	1.0
Gzip_pack	25	1.8	13	0.9
Group 4	19	1.4	8	0.6
Group 3	85	6.1	12	0.8
JBIG2	14	1.0	5	0.4
Rectangular	29	2.1	20	1.5

Figure 5 shows the communication energy consumption and its standard deviation for all the considered compression methods. It must be observed in Figure 5 that the standard deviation in the communication energy consumption for both the JBIG2 and CCITT Group 4 are the lowest amongst all the analyzed compression methods. It is also clear from Figure 5 that the communication energy consumption for both Gzip and CCITT Group 3 are higher when compared to all the other methods. JBIG2 and CCITT Group 4 have many good features such as resilience to the increase in the number of objects, increase in the size of the objects and to the different shapes of the objects. Based on these observations, it is possible to generalize that CCITT Group4, JBIG2 and Gzip_pack are good candidates for use in embedded applications of WVSN.

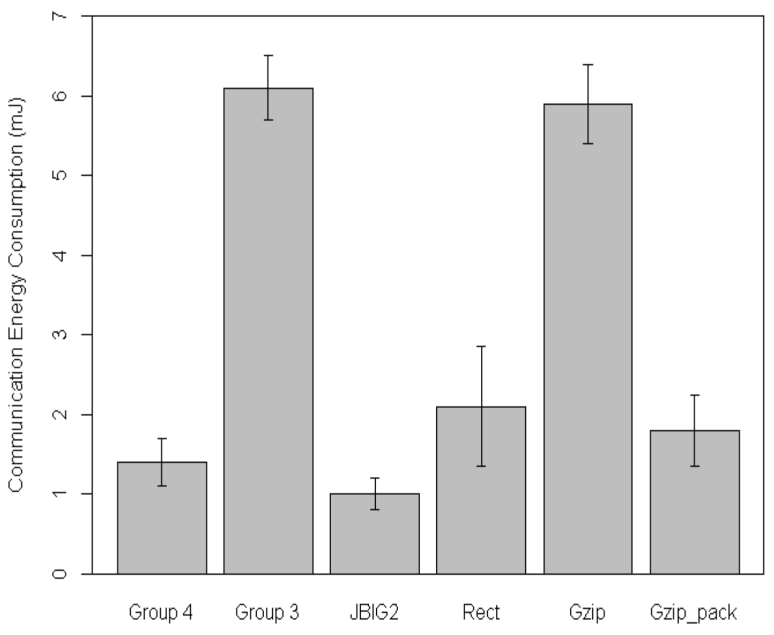

Figure 5: Communication energy consumption for six compression methods.

6. CONCLUSION

In this paper, we have analysed the effect of various features, such as the shape, size and number of objects in the bi-level images, on the compression efficiency of six different compression methods. The effect of increasing the size and number of objects in the input images was explored and the result is an increase in the size of the compressed file for all the compression methods. Some methods showed extreme level sensitivity for these features such as the Rectangular compression method. The sizes of the compressed files from Gzip, Rectangular and CCITT Group 3 compression methods are larger than those for the other three methods and because of this their communication energy consumptions are high. Additionally, the Rectangular compression method is extremely sensitive to the different features of the objects and its execution time is the highest of the analysed methods. Due to these characteristics the Rectangular compression method is not a suitable choice for remote applications of WVSN. CCITT Group 3 and Gzip are also out of race because of their high communication energies and poor resilience to the varying object features. The size of the compressed files from JBIG2 is lower than for all the other methods and it is highly resilient to different variations in the features of the objects. The processing time of JBIG2 is high for the software implementation but its communication energy consumption is the lowest of all the considered compression methods. The hardware implementation of JBIG2 could be faster. Although the Gzip_pack is moderately resilient to different variations in the objects features such as size, number and shape, its compression efficiency is high and its execution time is low. Hence, the conclusion drawn is that CCITT Group 4, JBIG2 and Gzip_pack are the most suitable bi-level compression methods for remote applications of WVSN.

REFERENCES

[1] K. Khursheed, M. Imran, M. O'Nils and N. Lawal, "Exploration of Local and Central Processing for a Wireless Camera Based Sensor Node," IEEE Int'l. Conf. Signal Elec. Syst., Gliwice, Poland, Sept. 7-10, 2010.

[2] K. Khursheed, M. Imran, M. O'Nils and N. Lawal, 2010. "Exploration of Target Architecture for a Wireless Camera Based Sensor Node," 28th Norchip Conference 15-16 November, Tampere, FINLAND.

[3] K. Khursheed, M. Imran, A.W. Malik, M. O'Nils, N. Lawal, T. Benny, 2011. "Exploration of Tasks Partitioning Between Hardware Software and Locality for a Wireless Camera Based Vision Sensor Node," Proc. of the Int'l Symp. on Parallel Computing in Electrical Engineering (PARELEC'2011).

[4] N. S. Jayant and P. Noll, "Digital Coding of Waveforms," Englewood Cliffs, NJ: Prentice-Hall, 1984.

[5] M. Aoki, "Rectangular region coding for binary image data compression,'' Pattern Recognition, vol. 11, pp. 297-312, 1979.

[6] R. Hunter and A. Robinson, “International digital facsimile coding standards,” proc, IEEE, vol, 68, pp, 856867, July 1980.

[7] I. Song and S. A. Kassam, “A new noiseless coding technique for binary images,” IEEE Trans. Acoust., Speech, Signal Processing. vol. 34, pp. 944-951, Aug. 1986.

[8] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” IEEE Trans. on Information Theory, vol. IT-23, no. 3, May 1977.

[9] S. A. Mohamed and M. M. Fahmy, “Binary Image Compression Using Efficient Partitioning into Rectangular Regions,” IEEE Trans. on Comm., vol. 43, no. 5, May 1995.

[10] J. Rissanen and G. G. Longdon, “Arithmetic coding,” IBM J. Res. Develop vol 23, pp 149-162, 1979.

[11] F. Ono, W. Rucklidge, R. Arps and C. Constantinescu. “ JBIG2- The ultimate bi-level image coding standard,” 0-7803-6297-7/00/$10.00 0 2000 IEEE.

[12] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr. and R. B. Arps, “An overview of the basic principle of the Q-Coder adaptive binary arithmetic coder,” IBM 1. Res. and Develop. Vol. 32 no. 6 page 717, November 1988.

[13] G. G. Langdon Jr. and J.,J. Rissanen, “Compression of black-white images with arithmetic coding," IEEE Trans. Communications, vol. 29(6), pp. 858-867, June 1981.

[14] A. Said, “Comparative Analysis of Arithmetic Coding Computational Complexity,” Results of this paper were presented as a poster at IEEE Data Compression Conference, 23-25 March 2004, Snowbird, Utah, USA.

[15] http://www.libtiff.org/libtiff.html

[16] http://tpgit.github.com/UnOfficialLeptDocs/leptonica/index.html

[17] F. Linnarsson, C. Peng and B. Oelmann, “Analysis of IEEE 802.15.4 standard for a wireless closed loop control system for heavy duty cranes”, In IEEE 2nd Int’l Symp. On Industrial Embedded Systems. - SIES'2007.

[18] http://www.gzip.org/

[19] G.G. Langdon, ” Probabilistic and Q-Coder algorithms for binary source adaptation,” In Proc. Data Compression Conference, J.A Storer and J.H. Reif eds., Snowbird, Uttah, Apr. 8-11,1991,13-22.

[20]. G.G. Langdon and J., Rissanen, “ Compression of black and white images with arithmetic coding,” IEEE Trans. Comm. COM-29(1981), 858-867.

[21] P.G. Howard and J. Vitter, “Arithmetic coding for data compression,” Proc. of IEEE, vol 82, no 6, 1994.

[22] http://www.itu.int/itu-t/recommendations/index.aspx?ser=T

[23] R.B. Arps, T.K. Truong, “Comparison of international standards for lossless still image compression,” Proceedings of the IEEE, vol 82, no 6, June 1994, 889 - 899.

[24]S.R. Kodituwakku, U.S., Amarasinghe, “Comparison of lossless data compression algorithms. for text,” Indian journal of computer science and engineering vol 1, no 4, 416-425.

[25] J. Kivijärvi, T., Ojala, T., Kaukoranta, A., Kuba, L., Nyul, O., Nevalainen, ”A comparison of lossless compression methods for medical images,” Computerized medical imaging graphics 1998; 22: 323-339.