N89-22341

A ROBUST COMPRESSION SYSTEM FOR LOW BIT RATE TELEMETRY - TEST RESULTS WITH LUNAR DATA

Khalid Sayood and Martin C. Rost
Department of Electrical Engineering
University of Nebraska

PROBLEM STATEMENT

The output of a Gamma Ray detector is quantized using a 14 bit A/D converter. The number of each of the 2^{14} or 16,384 levels occurring in a 30 second interval is counted. In effect, a histogram of the gamma ray events is obtained with 16,384 bins. The contents of these bins are to be encoded without distortion and transmitted at a rate less than or equal to 600 bits per second. Thus the contents of the 16,384 bins are to encoded using 18000 bits. The encoder should be simple to implement and require only a minimal amount of buffering.

PROPOSED SYSTEM

Encoder

The contents of the bins are treated as a sequence for purposes of encoding. The proposed system encoder can be divided into two stages (three if a Huffman coding option is used. See Figure 1.) The first stage is a leaky differencer whose input/output relationship is given by

$$z_n = x_n - [\, a x_{n-1} \,]$$

where [t] is the largest integer less than or equal to t. The reason for using a leaky differencer is to allow the effect of errors to die out with time.

The output of the differencer forms the input for the second stage which is a modified runlength encoder. The encoder codebook contains six different types of symbols.

Mn - symbol used to represent negative differencer output values, for example, the differencer output values -1, -2,...,-n, are represented by the symbols M1, M2,...,Mn, respectively.

Pn - symbols used to represent positive differencer values, they are coded similar to the Mn symbols. Thus a differencer output value of +3 would be represented by the symbol P3.

Zn - symbols used to represent string of zeros of length n. Since the number of Z-symbols is kept small, these symbols represent "short" string of zeros (0-strings), while the S0- and S1-symbols to be introduced later represent "long" 0-strings.

BR - In the encoding scheme that follows, there will sometimes be a need to specify the end of a sequence. The BR or break symbol is used for this purpose.

S0XX - symbol used to represent long 0-strings. The S0 symbol indicates that a 0-string is being represented while X stands for a four bit word. XX is thus an eight bit word specifying the length of the 0-string.

S1XX - symbol used to represent long 0-strings that are followed by a 1. It is constructed in the same manner as the S0XX symbol.

Each symbol, Mn, Pn, Zn, BR, S0, and S1 is represented by a four bit word. The number of symbols in the encoder codebook is o(M)+o(P)+o(Z)+3 where o(M), o(P), and o(Z) are, respectively, the number of negative source symbols, positive source symbols, and short 0-strings symbols to be channel coded. As each symbol is represented by 4 bits, a total of sixteen encoder symbols are possible. In our coding scheme, o(M) is set to 2, o(Z) to 6, and o(P) to 5.

This means that if the differential output is -1, -2, 1, 2, 3, 4, 5 or a string of zeros of length five or less, it can be represented by a single symbol. What if the differential output is a positive value larger than five or a negative value less than -2? In such cases the largest (in magnitude) Mn or Pn symbol is used as a concatenation symbol. As an example, consider encoding the value 18.

Since o(P) is 5, the largest positive value that can be coded with a single symbol is 5. If P5 is also used as a concatenation symbol, larger source values can be coded. In this case, 18 can be coded as P5 P5 P5 P3. The receiver accumulates a total for all the P5 symbols consecutively received until a non-P5 symbol is received. This symbol is used to complete the current source value. In this case, P3 indicates the source value is 18.

In the case where the source value is a multiple of the maximum P-symbol value some confusion can occur in the decoding process. Consider the coding of the source values 10 followed by 8. In this case, four source symbols are required to code these values but, the receiver decodes them as a 18. To overcome this problem the break symbol (BR) is used. This symbol carries no data value but, is used by the receiver to prematurely stop the accumulation of P-symbols. Specifically, 10 and 8 are coded as P5 P5 BR P5 P3. The receiver stops constructing the first source value when the BR is encountered and start constructing the next with the following P5 symbol.

If a source value to coded is negative, the above procedure is used with the allowed M-symbols along with the BR symbol to prevent incorrect receiver decoding. For example, -3 would be encoded as M2M1 and -4 would be encoded as M2M2BR.
In this particular application, the tails of a given signal frame contain long runs of zeros that are separated by non-zero data values. It is very likely that these 0-string separators take the value 1. Thus, it is beneficial to code these runs with one of the following two symbols, each of which is three code words in length:

S0 x y a 0-string of length xy (base 16).

S1 x y a 0-string of length xy (base 16) followed a 1.

For example, the symbol, S0 4 0, represents a string of 64 0s, and the symbol, S1 4 0, represents a string of 64 0s followed by a 1. If the separating data value is not 1, then additional source symbols follow the S0 symbol to complete the description of its value. The maximum length of 0-string that can be coded with this type symbol is 255 (FF base 16). If a string of length greater than 255 is encountered, a concatenation rule must be applied.

Since the symbols S0 0 0 and S1 0 0 are not assigned, they are used as 0-string concatenation symbols. They are used to indicate the fact that a 0-string is to be built whose length is greater than 255. Each time one of these symbols is used it is assumed that a 0-string of length greater than 255 is being coded, and additional information is to be provided on its length by the following symbols. A 0-string is terminated if the last S0-symbol indicates a length value other than 00 for xy.

For example, if a 0-string of length 300 is followed by a 1, two source symbols (six channel words) are required to code the string: S1 0 0 S1 2 D. The value for xy of the first symbol is 00, so the 0-string is continued using the following S1-symbol(s). In this way, 0-

strings of arbitrary length can be constructed by concatenating as many S1 0 0 symbols as needed to bring the overall reconstructed 0-string length to within 255 0s of its full length. The final S1-symbol in such a series which does not have a 0 0 length indicator terminates the 0-string concatenation process. Since the S1 symbol is being used this 0-string is automatically followed by a 1. Consider coding a 0-string of length 300 that is followed by a -1. Two S0-symbols (six channel words) are required to code the 0-string, and one M-symbol (one channel word) is required to code the -1: S0 0 0 S0 2 D M1 for a total of seven channel words.

Since the long runlength symbols require three channel words each, an excessive amount of channel capacity can be wasted when coding short runs of 0s. As a consequence, a group of short run symbols that use only one channel word each are used to alleviate this problem. The identifier for these symbols is Zn (where n represents the length of the 0-string). For example, a run of 5 0s is represented by the symbol Z5. The coding length of a short 0-string using Zn symbols only improves the overall coding rate if the short 0-string is coded with fewer channel bits when using the Z-symbols instead of the S0- and S1-symbols.

Consider the following example for coding a string of 10 0s. Since o(Z) is 6, to code this 0-string using Z-symbols takes two channel words: Z6 Z4. But, when coded using an S0-symbol it takes three channel words to code this 0-string: S0 0 A. Therefore, the Z-symbol coding is more channel efficient. Since an S0- (or S1-) symbol always require three channel words, the only way to guarantee that short 0-strings are coded efficiently is to set the maximum number of short Z-symbols in a single 0-string coding to two. Thus, for an o(Z) of 6, the maximum 0-string length to be Z-symbol coded is 12.

The encoder described above has two main characteristics. First, it has been designed for the specific task noted in the problem statement. No claims are made regarding its suitability for other

tasks. The second characteristic is its simplicity. The encoding operation requires a very small amount of computation. Furthermore, the onboard memory requirements for buffering are minimal.

If Huffman coding is to be used, the final stage of the encoder is a Huffman coder. This will, of course, increase the complexity of the encoder and may make the system more vulnerable to channel errors. Therefore, if at all possible we will avoid using a Huffman coder.

Decoder

The decoder for the proposed system consists of three stages. The first stage of proposed system decoder is maximum A Priori Probability (MAP) receiver[6]. The MAP receiver design is based on the assumption that the output of the encoder contains dependencies.

The MAP design criterion can be formally stated as follows: For a discrete memoryless channel (DMC), let the channel input alphabet be denoted by $A = \{a_0, a_1, \ldots, a_{M-1}\}$, and the channel input and output sequences by $Y = \{y_0, y_1, \ldots, y_{L-1}\}$ and $\hat{Y} = \{\hat{y}_0, \hat{y}_1, \ldots, \hat{y}_{L-1}\}$, respectively. If $A = \{A_i\}$ is the set of sequences $A_i = \{\alpha_{i,0}, \alpha_{i,1}, \ldots, \alpha_{i,L-1}\}, \alpha_{i,k} \epsilon A$, then the optimum receiver (in the sense of maximizing the probability of making a correct decision) maximizes $P[C]$, where

$$P[C] = \sum_{A_i} P[C|\hat{Y}]P[\hat{Y}].$$

This in turn implies that the optimum receiver maximizes $P[C|\hat{Y}]$. When the receiver selects the output to be A_k, then $P[C|\hat{Y}] = P[Y = A_k|\hat{Y}]$. Thus, the optimum receiver selects the sequence A_k such that

$$P[Y = A_k|\hat{Y}] \geq P[Y = A_i|\hat{Y}] \ \nabla_i.$$

When the channel input sequence is independent, this simplifies to the standard MAP receiver[6]. Under conditions where this is not true, the receiver becomes a sequence estimator which maximizes the path

metric. $\Sigma \log P(y_i|\hat{y}_i, y_{i-1})$ [5]. The path metric can be computed for a particular system by rewriting it using the following relationship[4].

$$P[y_i = a_j|\hat{y}_i = a_n, y_{i-1} = a_m] = \frac{P[\hat{y}_i = a_n|y_i = a_j]P[y_i = a_j|y_{i-1} = a_m]}{\Sigma_l \, P[y_i = a_l|y_{i-1} = a_m]P[\hat{y}_i = a_n|y_i = a_l]}$$

Notice that the right hand side consists of two sets of conditional probabilities $\{P[\hat{y}_i|y_i]\}$ and $\{P[y_i|y_{i-1}]\}$. The first set of conditional probabilities are the channel transition probabilities while the second depend only on the encoder output. The two are combined according to the above relationship to construct an $M \times M \times M$ lookup table for use in decoding. The structure of the MAP receiver is that of the Viterbi decoder[4,5].

The second stage of the decoder is the inverse operation of the modified run-length encoder. The operation of this stage has already been described in the previous section. The final stage of the decoder is the inverse of the differential operation with an input output relationship

$$x_n = z_n + [ax_{n-1}]$$

RESULTS

In this section we present results obtained by using the proposed system of the previous section. The data used was provided by Ms.\ M. Mingarelli-Armbruster of the Goddard Space Flight Center. This data was generated according to a Poisson distribution where the Poisson parameter was obtained from ten hours of lunar data. Both noisy and noiseless channel performance of the proposed system were examined via Monte-Carlo simulation. A total of twenty, 30-second intervals were used in the tests. The performance was compared with the Rice algorithm[1-3].

Before proceeding with the results, some caveats are in order. First,

the name Rice algorithm is a misnomer. What is presented[1-3] is not an algorithm but an approach. In this approach, a suite of algorithms is used to encode sections of the data, and the most efficient algorithm for that particular section of data is selected. In this way, data with very different statistical profiles can be accommodated. Thus what is presented[1-3] could more correctly be called the Rice Universal Coding Approach (RUCA). What we compare against here are algorithms presented[1-3] as examples of the RUCA. These algorithms were constructed for use in very general situations. As opposed to this, the particular algorithm presented here has been designed for a specific task. A final observation is that the encoder presented in this paper could very easily be used as the first stage of the RUCA. However, this would result in a rather complex encoder and substantial increase in the need for onboard memory over the proposed design. Therefore, if the algorithm presented in the previous section satisfies the requirements in terms of rate and robustness, such a step would be undesirable.

The results of the tests with both algorithms are presented in Table 1 and Table 2. The number of bits required to code twenty thirty-second intervals and the average rate needed for both algorithms is presented in Table 1. The second and third columns contain the total number of bits and the rate when the Rice algorithm is used. The average rate over twenty intervals is 719 bits per second. Columns three to six present the results obtained by using the proposed algorithm. The first two columns contain the results for the case where the Huffman coder was not used while the last two columns contain the results for when the Huffman coder formed the last stage of the encoder. The rate without the Huffman coder averaged over twenty intervals is 595 bits per second while the average rate when the Huffman coder is used is 522 bits per second. These results indicate that the proposed system will satisfy the specifications (coding rate below 600 bits per second) both when the Huffman coder is used and when it is not. As both systems meet the target and as the inclusion of the Huffman coder increases both the complexity and the vulnerability of the system to

channel noise, we elected to use the system without the Huffman coder.

Table 2 provides the performance of the algorithms under noisy channel conditions. Three performance measures are used, namely, mean squared error (MSE), mean absolute error (MAE), and the number of decoded values which are in error. Note the very large difference between the performance of the Rice algorithm and the proposed algorithm. Also, the proposed algorithm maintains a robust performance at extremely high error rates. In fact, under even highly adverse conditions the mean squared error is almost constant, and the number of erroneous decoded values is about 25% of the total. However, the performance of the algorithms at high error rates may be irrelevant in this particular situation. The reason being that the transmitted data will be well protected by a channel coding scheme consisting of a Reed-Solomon coder followed by a convolutional coder. This combination is expected to keep the average probability of error on the coded channel below 9×10^{-6}.

Finally, we examine the relative complexity and buffer requirements for the two algorithms. The proposed algorithm can be easily realized with a simple program implemented using a microprocessor. Based on the memory requirements for the simulation program used in this study, the memory needed for actual implementation should be about 1 K. The only time buffering may be required is when a large differencer output is encountered, and the encoder has to generate several channel symbols for one input. Depending on the way the entire system is implemented, the buffer requirements could range from a single symbol buffer to perhaps a sixteen symbol buffer.

As opposed to this, the Rice algorithm by its very nature, being a universal coding algorithm, is quite complex. Each block of data is encoded using a number of candidate algorithms; the algorithm which provides the most efficient encoding is then selected. Each of the candidate algorithms is itself relatively complex though some very ingenious techniques are used to make subunits of one algorithm common

to several candidate algorithms. Because several passes are required to do the encoding, the buffering requirements for this approach are substantial.

These differences in complexity are very natural based on the different objectives of the two algorithms. The proposed system is designed for a very specific situation while the Rice algorithm is designed to handle general situations.

TABLE 1

Coding rates with the Rice Algorithm and the Proposed Algorithm, (HC) denotes the results for the case where the Huffman Coder was used.

	RICE ALGORITHM		PROPOSED ALGORITHM			
INTERVAL	TOTAL BITS	RATE	TOTAL BITS	RATE	TOTAL BITS (HC)	RATE (HC)
1	21,647	721.6	17,832	594.4	15,733	524.4
2	21,385	712.8	17.528	584.3	15,345	511.5
3	21,530	717.7	17,784	592.8	15,520	517.3
4	21,562	718.7	17,840	594.7	15,691	523.0
5	21,666	722.2	18,144	604.8	15,883	529.4
6	21,424	714.1	17,504	583.5	15,457	515.2
7	21,841	728.0	18,048	601.6	15,882	529.4
8	21,630	721.0	18,096	603.2	15,907	530.2
9	21,719	723.9	18,132	604.4	15,843	528.1
10	21,568	718.9	18,096	603.2	15,695	523.2
11	21,308	710.3	17,604	586.8	15,438	514.6
12	21,509	716.9	17,728	590.9	15,580	519.3
13	21.633	721.1	17,780	592.7	15,581	519.4
14	21,822	727.4	18,016	600.5	15,913	530.4
15	21,296	709.8	17,564	585.4	15,361	512.0
16	21,701	723.4	17,956	598.5	15,872	529.1
17	21,058	701.9	17,296	576.5	15,139	504.6
18	21,312	710.4	17,688	589.6	15,449	514.9
19	21,713	723.8	18,160	605.3	16,033	534.4
20	21,888	729.6	18,292	609.7	16,125	537.5
OVERALL AVERAGE		718.7		595.1		522.4

TABLE 2

Performance of the algorithms under noisy channel conditions.

	RICE ALGORITHM		
PROBABILITY OF ERROR	MEAN SQUARED ERROR	MEAN ABSOLUTE ERROR	# OF DECODED ERRORS
10^{-6}	0.0760	0.023	140
10^{-5}	4.07	0.45	1,908
10^{-4}	31.49	3.14	10,177
10^{-3}	479.22	16.03	15,658
10^{-2}	8,562.87	76.75	16,189

	PROPOSED ALGORITHM		
PROBABILITY OF ERROR	MEAN SQUARED ERROR	MEAN ABSOLUTE ERROR	# OF DECODED ERROR
10^{-6}	2.4×10^{-5}	1.2×10^{-5}	1
10^{-5}	0.026	0.016	218
10^{-4}	0.17	0.14	1,287
10^{-3}	0.78	0.28	2,944
10^{-2}	6.81	0.71	3,765

SUMMARY AND CONCLUSIONS

We have presented a robust noiseless encoding scheme for encoding the gamma ray spectroscopy data. The encoding algorithm is simple to implement and has minimal buffering requirements. The decoder contains error correcting capability in the form of a MAP receiver. While the MAP receiver adds some complexity, this is limited to the decoder. Nothing additional is needed at the encoder side for its functioning.

ACKNOWLEDGMENT

This work was supported by a grant from the Goddard Space Flight Center, Greenbelt, Maryland, under Grant NAG5-916.

REFERENCES

1) R. F. Rice, "Practical Universal Noiseless Coding," 1979 SPIE Symposium Proceedings, Vol.\ 207, San Diego, CA, August 1979, pp. 247-267.

2) R. F. Rice, "Some Practical Universal Noiseless Coding Techniques," JPL Publication 79-22, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, March 15, 1979.

3) R. F. Rice and Jun-Ji Lee, "Some Practical Universal Noiseless Coding Techniques, Part II," JPL Publication 83-17, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, March 1, 1983.

4) K. Sayood and J. C. Borkenhagen, "Use of Residual Redundancy in the Design of Joint Source/Channel Coders," Submitted to IEEE Trans.\ Commun.

5) K. Sayood and J. D. Gibson, "Maximum Aposteriori Joint Source/Channel Coding," Proceedings of the 22nd Annual Conference on Information Sciences and Systems, Princeton, New Jersey, March 1988.

6) J. M. Wozencraft and I. M. Jacobs, "Principles of Communication Engineering. John Wiley and Sons, Inc., New York, 1965.

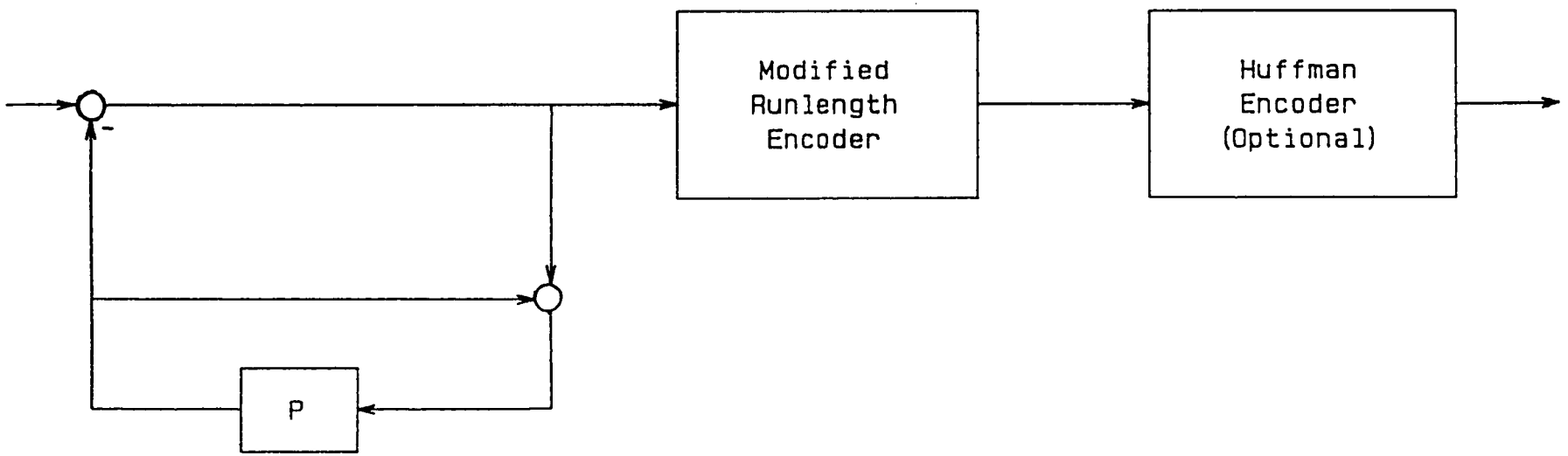

Figure 1. Proposed Encoder