Development of Efficient Decoding Technique for Convolutionaly Encoded Telemetry Data

Namratha M[1], Pradeep[2]

[1,2] (Dept. of information science, PESIT/visvesvaraya technological university, India)

Abstract

Telemetry system of spacecraft is meant for radioing information from a spacecraft to the ground. Telemetry is typically a mixture of payload or housekeeping health data. Engineering or health data is composed of a wide range of measurements, from switch positions and subsystem states to voltages, temperatures and pressures. Telemetry may be transmitted in real time, or it may be written to a data storage device until transmission is feasible. As the spacecraft health monitoring is very important to maintain the spacecraft in condition, Telemetry system becomes mission critical system in the spacecraft. Packet Telemetry and Telemetry Channel Coding services provide to the user reliable and transparent delivery of telemetry information. With the error detecting and correcting capability of the channel code chosen, errors which occur as a result of the physical transmission process may be detected and corrected by the receiving entity. The implementation of convolution encoder is simple, this means these encoders are small and consumes less power on-board spacecraft and thus it has a good attribute as a spacecraft hardware. The proposed paper aims at decoding the convolution encoded system. There are various approaches to decode and broadly they are classified as

- Sequential decoding
- Maximum likelihood decoding based on Viterbi decoding

The development of an algorithm, implementation of the same and testing the same with simulated convolution encoded data stream is planned as part of this paper.

Keywords: Branch metrics, Convolution encoder, Forward error correction, Path metrics, Survivor paths, Trellis diagram, Viberti algorithm

I. INTRODUCTION

The probability of bit error P_e[1] in a digital transmission can be improved to attain acceptable reliability by increasing E_b/N_o. As E_b/N_o is directly proportional to $C/N_{0(}$(the ratio of the average carrier power to the noise power density) and the desired E_b/N_0 can be achieved by increasing the transmitted power and/or reducing the system noise temperature (to reduce N_0). Both these measures in a spacecraft are limited by cost, size, power and thermal constraints.. Channel coding is an acceptable alternative. In fact for a fixed E_b/N_0, coding is the method of lowering the BER. Channel Coding performs two functions, error detection and error correction. What is termed as forward error correction (FEC) allows errors to be corrected without the need for retransmission.In order to overcome this limitation convolutional coding and decoding method can be used. Convolutional coding is particularly suited to space and telemetry systems that require simple and small encoders and that obtain excellent coding gain by using sophisticated decoders at the ground station.
The functions of this software product are as follows:

- Configure the software based on the user requirement (like constraint length).
- Encodes data either represented in binary or hexadecimal format
- Decodes the data to recover the original data.

The system at present is configured for encoding a block of data obtained . It cannot be used to encode a continuous stream of data. In order to do it the system must be configured accordingly. It is assumed that the decoder is synchronized with the stream of incoming data .If the decoder doesnot know which of the n symbols in block initiates a branch of the trellis , then the data will be decoded with a very large number of errors .It also assumed that encoded data is not modified such that some part of it is missing. It contains all the software requirements to level of detail sufficient to enable designers to design a system to satisfy those requirements , and testers to test that the system satisfies those requirements.

II. DESIGN CONSIDERATIONS

- Design should allow for users to specify the two generator polynomials based on which the data is encoded.
- Should also for request constraint length
- Other options such as inversion of the output if required should be provided
- The data to be encoded in binary or hexadecimal format is generated
- The encoded data in binary format
- If any of the input like generator polynomial is missing error message should be generated.

2.1 System Design:

The system that implements forward error correction mainly consists of a channel encoder which adds redundant information to the stream of input symbols in a way that allows errors which are in the channel to be corrected .This redundancy is provided in a structured way to provide error control capability .Because of the redundancy introduced by the channel coder, there must be more symbols at the output of the coder than the input .The input to the channel coder is referred to as the message symbols .The input may also be referred to as information symbols .The stream that consists of the coded message and injected noise is input to the Viterbi decoder. Within the decoder, a Metric Update kernel is performed, which produces two streams – a state metric stream, which contains the accumulated state metrics for all delay states, and a transition stream, which contains the optimal path chosen for each delay state. These two streams are passed to a trace back stream, which traverses the state metric stream and employs the transition stream to find the optimal path through the Trellis.

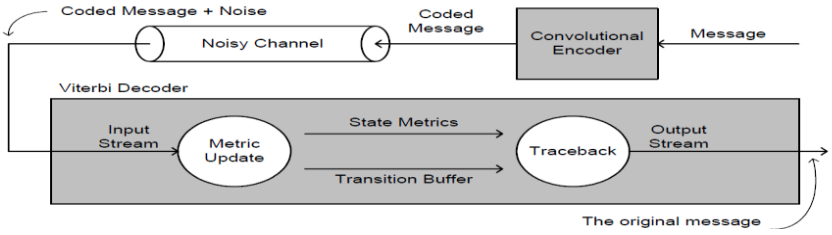


Fig 1: Simplified block of forward error correction system.

Like any error-correcting code, a convolutional code works by adding some structured redundant information to the user's data and then correcting errors using this information. A convolutional encoder is a linear system .A binary convolutional encoder can be represented as a shift register. The outputs of the encoder are modulo 2 sums of the values in the certain register's cells. The input to the encoder is either the un encoded sequence (for non-recursive codes) or the un encoded sequence added with the values of some register's cells (for recursive codes). Convolutional codes are frequently used to correct errors in noisy channels. They have rather good correcting capability and perform well even on very bad channels (with error probabilities of about 10^{-3}).

2.2 Convolutional Encoder

A binary convolutional encoder can be represented as a shift register. The encoder for a binary (2,1,4) code is shown in figure. Note that encoder consists of an m=3 stage shift register together with an n=3 modulo 2 adders and a multiplexer for serializing the encode outputs. The mod-2 adders can be implemented as EXCLUSIVE-OR gates. Since The mod-2 addition is a linear operation. The encoder[2] is a linear feed forward shift register. All the convolutional encoders can be implemented using a linear feed forward shift register of this type.

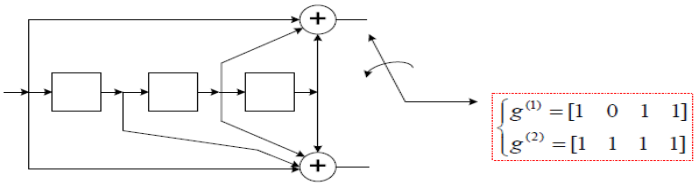


Fig 2: Convolution Encoder using shift registers

The information sequence $u=(u_0,u_1,u_2\ldots.)$ enters the encoder one bit at a time. Since the encoder is a linear system ,the two encoder output sequences $v^{(1)} =(v_0^{(1)} , v_1^{(1)} , v_2^{(1)} \ldots.)$ and $v^{(2)} =(v_0^{(2)} , v_1^{(2)} , v_2^{(2)} \ldots.)$ can be obtained as the convolution of the

input sequence with the two encoder "impulse" responses .The impulse responses are obtained by letting u=(1 0 0…) and observing output sequences. Since the encoder has m –time unit memory, the impulse responses can last at most m+1 time units, and are written $g^{(1)} =(g_0^{(1)}, g_1^{(1)} \ldots\ldots g_m^{(1)})$ and written $g^{(2)} =(g_0^{(2)}, g_1^{(2)} \ldots\ldots g_m^{(2)})$ .The impulse responses for the fig are

G(1)=1011 G(2)=1111

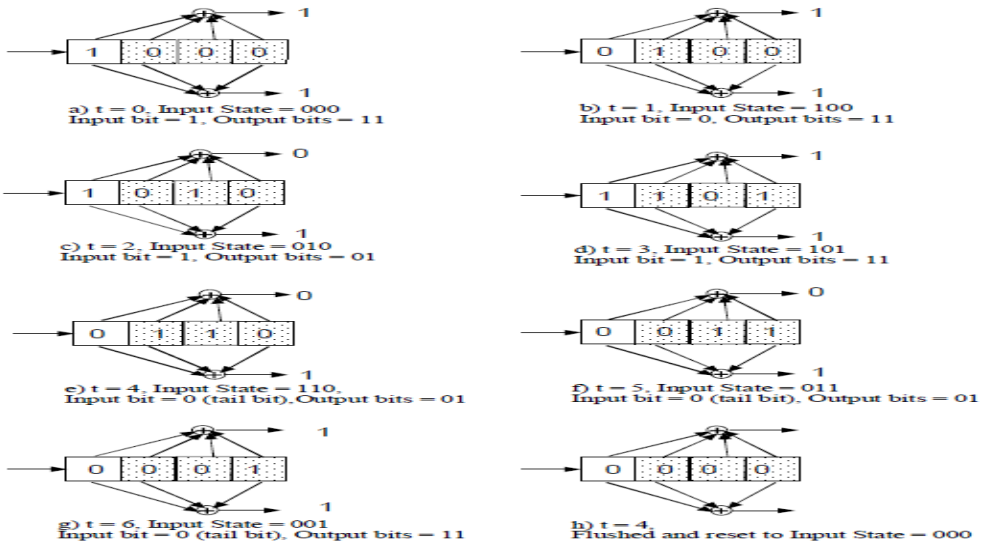
The impulse responses g(1) , g(2) are called generator sequences of the code. The encoding equations can be written as $v^{(1)} =u*g^{(1)}$ $v^{(2)} =u*g^{(2)}$ where * denotes discrete convolution and all operation are modulo-2.the convolution operation implies that for all l>=0,

$$v_l^{(j)} = \sum_{i=0}^{m} u_{l-i}\, g_i^j = u_l g_0^{(j)} + u_{l-1} g_1^{(j)} + \ldots\ldots\ldots\ldots\ldots + u_{l-m} g_m^{(j)} \quad \text{for } j=1,2..$$

After encoding,the two output sequences are multiplexed into a single sequence,called the code word,for transmission over channel

$V=(v_0^{(1)}\ v_0^{(2)}, v_1^{(1)}\ v_1^{(2)}, v_2^{(1)}\ v_2^{(2)}, \ldots)$

The encoding procedure is described here for sequence of 10 with the (2,1,4) code for the input sequence 1011 a) At time t = 0, we see that the initial state of the encoder is all zeros (the bits in the right most register positions). The input bit 1 causes two bits 11 to be output by a mod2 sum of all bits in the registers for the first bit and a mod2 sum of three bits for second output bit per the polynomial coefficients.b) At t = 1, the input bit 0 moves forward one register. The encoder is now in state 100. The output bits are now again 11 by the same procedure.c)At time t=2,the input bit 1 moves forward one register .the encoder is now in state 010.The output bits are produced similarly.At t=3,4 the same process continues the output produced is11 11 01 11.The entire process is summarized in the Fig 3 below.



A convolutional encoder[3] is often seen as a finite state machine. Each state corresponds to some value of the encoder's register. Given the input bit value, from a certain state the encoder can move to two other states. These state transitions constitute a diagram which is called a trellis diagram. A trellis diagram for the code on the Figure 2 is depicted on the Figure 3. A solid line corresponds to input 0, a dotted line – to input 1 (note that encoder states are designated in such a way that the rightmost bit is the newest one).

2.3 Trellis Diagram

Each path on the trellis diagram[4] corresponds to a valid sequence from the encoder's output. Conversely, any valid sequence from the encoder's output can be represented as a path on the trellis diagram. One of the possible paths is denoted as red (as an example). Each state transition on the diagram corresponds to a pair of output bits. There are only two allowed transitions for

every state, so there are two allowed pairs of output bits, and the two other pairs are forbidden. If an error occurs, it is very likely that the receiver will get a set of forbidden pairs, which don't constitute a path on the trellis diagram. So, the task of the decoder is to find a path on the trellis diagram which is the closest match to the received sequence.

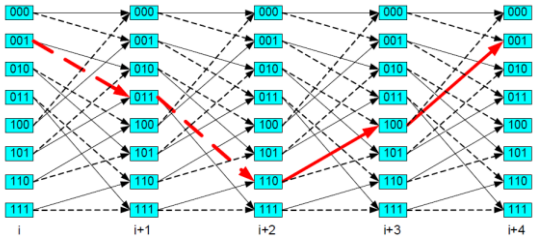


Fig 4: Trellis diagram

2.4 Viterbi algorithm

Viterbi algorithm[5] reconstructs the maximum-likelihood path given the input sequence. A soft decision decoder – a decoder receiving bits from the channel with some kind of reliability estimate. Three bits are usually sufficient for this task. Further increasing soft decision width will increase performance only slightly while considerably increasing computational difficulty. For example, if we use a 3-bit soft decision, then "000" is the strongest zero, "011" is a weakest zero, "100" is a weakest one and "111" is a strongest one.A hard decision decoder – a decoder which receives only bits from the channel (without any reliability estimate). A branch metric – a distance between the received pair of bits and one of the "ideal" pairs ("00","01", "10", "11").A path metric is a sum of metrics of all branches in the path.A meaning of distance in this context depends on the type of the decoder:

• for a hard decision decoder it is a Hamming distance, i.e. a number of differing bits;

• for a soft decision decoder it is an Euclidean distance.

In these terms, the maximum-likelihood path is a path with the minimal path metric. Thus the problem of decoding is equivalent to the problem of finding such a path.Let's suppose that for every possible encoder state we know a path with minimum metric[6] ending in this state. For any given encoder state there is two (and only two) states from which the encoder can move to that state, and for both of these transitions we know branch metrics. So, there are only two paths ending in any given state on the next step. One of them has higher metric, it is a survivor path.A Viterbi algorithm[7] consists of the following three major parts:

1. Branch metric calculation – calculation of a distance between the input pair of bits and the four possible "ideal" pairs ("00", "01", "10", "11").
2. Path metric calculation – for every encoder state, calculate a metric for the survivor path ending in this state (a survivor path is a path with the minimum metric).
3. Traceback[8] – this step is necessary for hardware implementations that don't store full information about the survivor paths, but store only one bit decision every time when one survivor path is selected from the two.

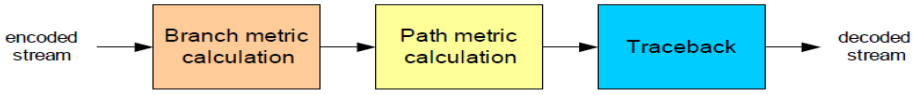


Fig. 5: Viterbi decoder data flow

Branch Metric Calculation:

Methods of branch metric calculation are different for hard decision and soft decision decoders.

For a *hard decision* decoder, a branch metric is a Hamming distance between the received pair of bits and the "ideal" pair. Therefore, a branch metric can take values of 0, 1 and 2. Thus for every input pair we have 4 branch metrics (one for each pair of "ideal" values).

For a *soft decision* decoder, a branch metric is measured using the Euclidean distance. Let x be the first received bit in the pair, y – the second, $x0$ and $y0$ – the "ideal" values. Then branch metric is $Mb=(x-x_0^2)+(y-y_0^2)$.

Furthermore, when we calculate 4 branch metric for a soft decision decoder, we don't actually need to know absolute metric values – only the difference between them makes sense. So, nothing will change if we subtract one value from the all four branch metrics:

$Mb=(x2-2\,x\,x_0+x_0^{\,2})+(y^2-2\,y\,y_0+y_0^{\,2})$;

$Mb^* = Mb - x^2 - y^2 = (x_0^2 - 2\,x\,x_0) + (y_0^2 - 2\,y\,y_0)$.

Path Metric Calculation:

Path metrics are calculated using a procedure called *ACS* (Add-Compare-Select). This procedure is repeated for every encoder state.

1. Add – for a given state, we know two states on the previous step which can move to this State , and the output bit pairs tha t correspond to these transitions. To calculate new path Metrics , we add the previous path metrics with the corresponding branch metrics.
2. Compare, select – we now have two paths, ending in a given state. One of them (with greater metric) is dropped.

As there are 2^{K-1} encoder states, we have 2^{K-1} survivor paths at any given time.

It is important that the difference between two survivor path metrics cannot exceed $\delta \log(K-1)$, where δ is a difference between maximum and minimum possible branch metrics.

The problem with path metrics is that they tend to grow constantly and will eventually overflow.

But, since the absolute values of path metric don't actually matter, and the difference between them is limited, a data type with a certain number of bits will be sufficient.

There are two ways of dealing with this problem:

1. Since the absolute values of path metric don't actually matter, we can at any time subtract an identical value from the metric of every path. It is usually done when *all* path metrics exceed a chosen threshold (in this case the threshold value is subtracted from every path metric). This method is simple, but not very efficient when implemented in hardware.
2. The second approach allows overflow, but uses a sufficient number of bits to be able to detect whether the overflow took place or not. The *compare* procedure must be modified in this case. The whole range of the data type's capacity is divided into 4 equal parts. If one path metric is in the 3-rd quarter, and the other – in the 0-th, then the overflow took place and the path inthe 3-rd quarter should be selected. In other cases an ordinary compare procedure is applied. This works, because a difference between path metrics can't exceed a threshold value, and the range of path variable is selected such that it is at least two times greater than the threshold.

Traceback:

It has been proven that all survivor paths merge after decoding a sufficiently large block of data (D in fig) i.e. they differ only in their endings and have the common beginning.

If we decode a continuous stream of data, we want our decoder to have finite latency. It is obvious that when some part of path at the beginning of the graph belongs to every survivor path, the decoded bits corresponding to this part can be sent to the output. Given the above statement, we can perform the decoding as follows:

1. Find the survivor paths for $N+D$ input pairs of bits.
2. *Trace back* from the end of any survivor paths to the beginning.
3. Send N bits to the output.
4. Find the survivor paths for another N pairs of input bits.
5. Go to step 2.

In these procedure[9] D is an important parameter called decoding depth. A decoding depth should be considerably large for quality decoding, no less then $5K$. Increasing D decreases the probability of a decoding error, but also increases latency.

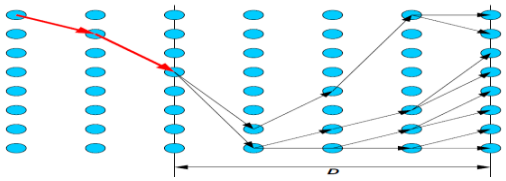


Fig. 6: Survivor paths graph example.

Blue circles denote encoder states. It can be seen that all survivor paths have a common beginning (red) and differ only in their endings.

Iii. Testing

A few of the sample test cases are as follows:

Serial No. of test case:	UTC-1
Module under test:	ConfigureCODEC
Description:	Configures the user input
Sample input:	User input
Expected Output:	it should configure the user input
Actual Output:	It is configuring the user input
Remarks:	Test is successful

Serial No. of test case:	UTC-2
Module under test:	CCSDS_Encoder
Description:	Encodes the input data
Sample input:	Data given by the user
Expected Output:	It should encode the data
Actual Output:	It is encoding the data
Remarks:	Test is successful

Serial No. of test case:	UTC-3
Module under test:	CCSDS_Decoder
Description:	Decodes the received data
Sample input:	Input from the encoder
Expected Output:	Decode the received data
Actual Output:	Decoding the received data
Remarks:	Test is successful

IV. Conclusion And Future Work

This paper attempted implementation of encoding and decoding of telemetry data. The major portion of the project is concentrated on C implementation on Windows platform which has been done successfully, experimentation with different input. In this application, the data is not a continuous stream and the noise is not considered . Further work on how to makethedata a continuous stream can be done. Noise should be simulated. Graphical representation of output from encoder and decoder.

REFERENCES

[1] http://www.radio-electronics.com/info/rf-technology-design/ber/bit-error-rate-tutorial-definition.php
[2] http://apogeelabs.com/pdffiles/convolutional_encoding.pdf
[3] http://www.comlab.hut.fi/opetus/333/2004_2005_slides/Convolutional_Coding_Viterbi_Algorithm.pdf
[4]http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=650373
[5] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=5443417&contentType=Conference+Publications
[6] http://www.1-core.com/library/comm/viterbi/
[7] http://www.ece.umd.edu/~tretter/enee722/ungerboeck_paper.pdf
[8]http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1450960&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1450960
[9]http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=410439&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D410439