Convolutional Codes

Victor Tomashevich, Advisor: Pavol Hanus

I. INTRODUCTION

The difference between block codes and convolutional codes is the encoding principle. In the block codes, the information bits are followed by the parity bits. In convolutional codes the information bits are spread along the sequence. That means that the convolutional codes map information to code bits not block wise, but sequentially convolve the sequence of information bits according to some rule.
The code is defined by the circuit, which consists of different number of shift registers allowing building different codes in terms of complexity.
Consider the following circuit (all registers are initially zero):

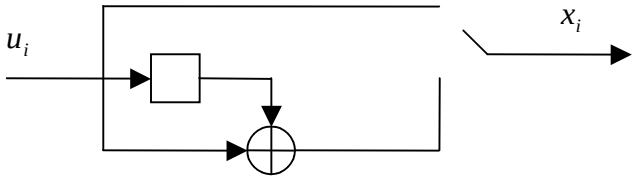


Fig.1. Encoding circuit of rate ½ code

The code bits $(x_0^{(1)}, x_1^{(1)}, \ldots)$ and $(x_0^{(2)}, x_1^{(2)}, \ldots)$ are obtained as follows:

$$x_i^{(1)} = u_i \text{ and } x_i^{(2)} = u_i + u_{i-1} \tag{1}$$

Then the codewords are generated:

$$\underline{x} = ((x_0^{(1)} x_0^{(2)}), (x_1^{(1)} x_1^{(2)}), \ldots) = (\underline{x}_0, \underline{x}_1, \ldots) \tag{2}$$

A convolutional encoder encodes K information bits to $N > K$ code bits in each time step. It is obvious, that for this particular code $K = 1$ and $N = 2$, giving the code a rate $R = 1/2$. The encoding procedure is not memoryless, as the code bits depend on the information bits encoded at past time steps. This is another big difference from block codes, as the block codes are memoryless.
The fact that the convolutional codes have memory allows them to operate well, when K and N are pretty small. The block codes have to have long block lengths (e.g., Reed-Solomon code used in CD drives has $N = 2048$), because they are memoryless and their performance improves with block length.

II. CONVOLUTIONAL CODES

A. Properties of convolutional codes [1]

- The convolutional code is linear: any linear combination of code bit sequences is again a valid code bit sequence. We can therefore specify the distance properties of the code by investigating only the weights of non-zero sequences.
- The encoding mapping $\underline{u} \to \underline{x}$ is bijective, i.e., each code bit sequence uniquely corresponds to an information bit sequence.
- Code bits generated at time step i are affected by information bits up to M time steps $i-1, i-2, \ldots i-M$ back in time, i.e., M is the maximal delay of information bits in the encoder.
- The code memory is the minimal number of shift registers required to construct an encoding circuit for the code. The code memory is at most $M \cdot K$.
- The constraint length is the overall number of information bits affecting code bits generated at time step i: $M \cdot K + K = (M+1) \cdot K$

- A convolutional code is systematic if the N code bits generated at time step i contain the K information bits

Let us consider some examples:

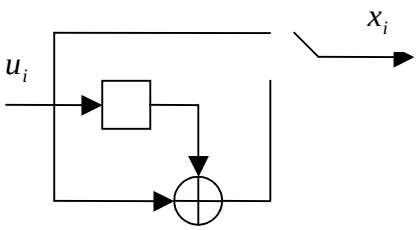


Fig.2. This rate $R = 1/2$ code has delay $M = 1$, memory 1, constraint length 2, and is systematic

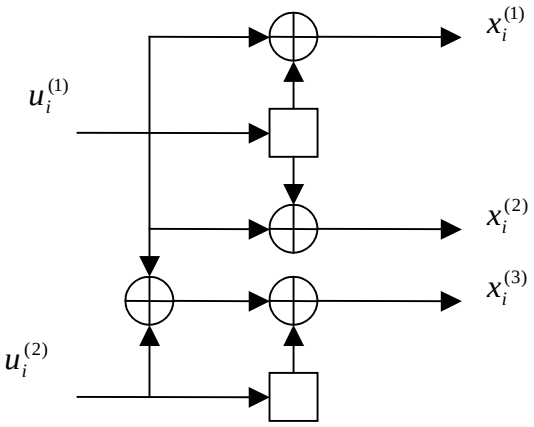


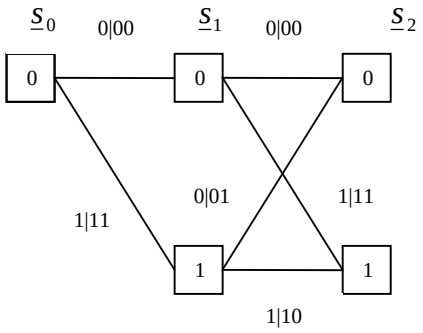
Fig.3. This rate $R = 2/3$ code has delay $M = 1$, memory 2, constraint length 4, and is not systematic

B. Descriptions of convolutional codes

There are multiple ways to describe convolutional codes, the main are [1]:

- Trellis
- Matrix description

First, we consider the description using a trellis. Trellis is a directed graph with at most S nodes, where S is the vector representing all the possible states of the shift registers at each time step i. In all the examples the rate $R = 1/2$ code described in Fig.1. is used.



operation of the encoding circuit:

- The trellis section has $2^{M \cdot K}$ nodes for each time step.
- It is obvious that $s_i = u_{i-1}$, and the codeword is obtained by $x_i^{(1)} = u_i$ and $x_i^{(2)} = u_i + s_i$.
- The branches are labeled with $\underline{u}_i \mid \underline{x}_i$. So we input a new information bit u_i and it is written into the shift register, thus $s_{i+1} = u_i$, and we get a codeword $\underline{x}_i$, generated with this transition.
- The trellis is growing exponentially with M.

As described in [1], we can define a convolutional code, using a generator matrix that describes the encoding function $\underline{u} \rightarrow \underline{x}$:

$$\underline{x} = \underline{u} \cdot \underline{G} \tag{3}$$

where

$$\underline{G} = \begin{pmatrix} \underline{G}_0 & \underline{G}_1 & \underline{G}_2 & \cdots & \underline{G}_M & & \\ & \underline{G}_0 & \underline{G}_1 & \underline{G}_2 & \cdots & \underline{G}_M & \\ & & \underline{G}_0 & \underline{G}_1 & \underline{G}_2 & \cdots & \underline{G}_M \\ & & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{4}$$

The $(K \times N)$ submatrices $\underline{G}_m, m = 0,1,\ldots,M$, with elements from $GF(2)$ specify how an information bit block $\underline{u}_{i-m}$, delayed m time steps, affects the code bit block $\underline{x}_i$:

$$\underline{x}_i = \sum_{m=0}^{M} \underline{u}_{i-m} \underline{G}_m, \forall i \tag{5}$$

For this code (in Fig.1.) we need to specify two submatrices, $\underline{G}_0$ and as the memory of this code is $M = 1$, also $\underline{G}_1$. The size of the submatrices must be $(K \times N)$, and as we have a rate $R = 1/2$ code, both submatrices will be of size 1×2.

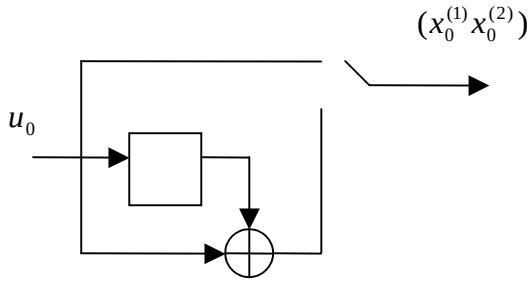
The matrix $\underline{G}_0$ governs how $\underline{u}_i$ affects $\underline{x}_i = (x_i^{(1)}, x_i^{(2)})$: as $\underline{u}_i$ affects the first bit of the codeword as well as the second bit of the codeword the matrix $\underline{G}_0 = (1 \quad 1)$.

The matrix $\underline{G}_1$ governs how $\underline{u}_{i-1}$ affects $\underline{x}_i$: as $\underline{u}_{i-1}$ affects only the second bit of the codeword but not the first one the matrix $\underline{G}_1 = (0 \quad 1)$.

For 3 information bit long sequence $\underline{u} = (u_0, u_1, u_2)$ using (3) we get

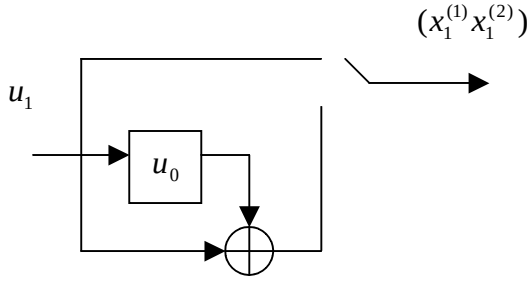
$$((x_0^{(1)} x_0^{(2)}), (x_1^{(1)} x_1^{(2)}), (x_2^{(1)} x_2^{(2)})) = (u_0, u_1, u_2) \cdot \underline{G} \tag{6}$$

As we have 3 information bits at the input and we consider 3 time steps the generator matrix $\underline{G}$ must be 3×3. The generator matrix is obtained considering the operation of the encoding circuit:



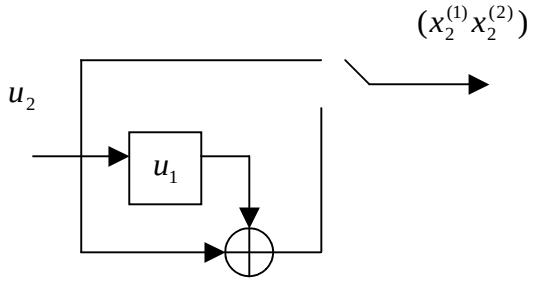
At time $t = 0$ we input u_0, as there is nothing in the register both code bits are affected by u_0. The generator matrix at this time step is:

$$\underline{G} = \begin{pmatrix} 11 & & \\ & & \\ & & \end{pmatrix}$$



At time $t = 1$ we input u_1. As u_0 is stored at the register it affects only the second bit of the code word, u_1 affects both bits of the codeword. The generator matrix at this time step is:

$$\underline{G} = \begin{pmatrix} 11 & 01 & \\ & 11 & \\ & & \end{pmatrix}$$



At time $t = 2$ we input u_2. As u_1 is stored at the register it affects only second bit of the generated codeword, u_2 affects both bits of the codeword and u_0 doesn't affect anything as it was already deleted from the register. Finally we obtain the whole generator matrix:

$$\underline{G} = \begin{pmatrix} 11 & 01 & \\ & 11 & 01 \\ & & 11 \end{pmatrix}$$

C. Puncturing of convolutional codes

The idea of puncturing is to delete some bits in the code bit sequence according to a fixed rule. In general the puncturing of a rate K/N code is defined using N puncturing vectors. Each table contains p bits, where p is the puncturing period. If a bit is 1 then the corresponding code bit is not deleted, if the bit is 0, the corresponding code bit is deleted. The N puncturing vectors are combined in a $N \times p$ puncturing matrix $\underline{P}$.

Consider code in Fig.1. , without puncturing, the information bit sequence $\underline{u} = (0,0,1,1,0)$ generates the (unpunctured) code bit sequence $\underline{x}_{NP} = (00,00,11,01,01)$. The sequence $\underline{x}_{NP}$ is punctured using a puncturing matrix:

$$\underline{P}_1 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

The puncturing period is 4. Using $\underline{P}_1$, 3 out of 4 code bits $x_i^{(1)}$ and 2 out of 4 code bits $x_i^{(2)}$ of the mother codes are used, the others are deleted. The rate of the punctured code is thus $R = 1/2 \cdot (4+4)/(3+2) = 4/5$ and $\underline{u}$ is encoded to $\underline{x} = (00,0X,1X,X1,01) = (00,0,1,1,01)$

The performance of the punctured code is worse than the performance of the mother code. The advantage of using puncturing is that all punctured codes can be decoded by a decoder that is able to decode the mother code, so only one decoder is needed. Using different puncturing schemes one can adapt to the channel, using the channel state information, send more redundancy, if the channel quality is bad and send less redundancy/more information if the channel quality is better.

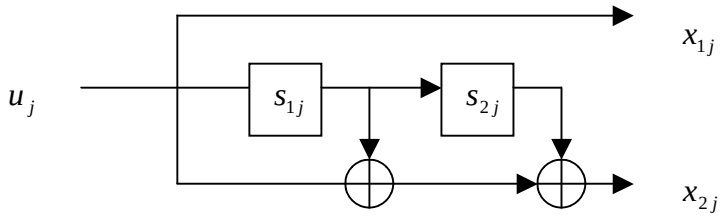
D. Decoding of convolutional codes

The very popular decoding algorithm for convolutional codes, used in GSM standard for instance, is the Viterbi algorithm. It uses the Maximum likelihood decoding principle. The Viterbi algorithm consists of the following steps for each time index [1]:

1. For all S state nodes at time step j, $j = 0,1,\ldots,L/N-1$:
   - Compute the metrics for each path of the trellis ending in the state node. The metric at the next state node is obtained by adding the path metric to the metric at the previous corresponding state node. The path metric is obtained by the following formula, : $\lambda_j = x_{1j} \cdot y_{1j} + x_{2j} \cdot y_{2j}$ (7)
     where x is the sent bit, y is the received bit
   - If the two paths merge, choose the one with the largest metric, the other one is discarded. If the metrics are equal, the survivor path is chosen by some fixed rule.

2. If the algorithm processed d trellis sections or more, choose the path with the largest metric, go back through the trellis and output the code bit sequence and the information bit sequence. The parameter d is the decision delay of the algorithm and specifies how many received symbols have to be processed until the first block of decoded bits is available. As a rule of thumb, the decision delay is often set to $5 \cdot M$.

Let us consider a following encoding circuit:



In Fig.4. the corresponding trellis is depicted:

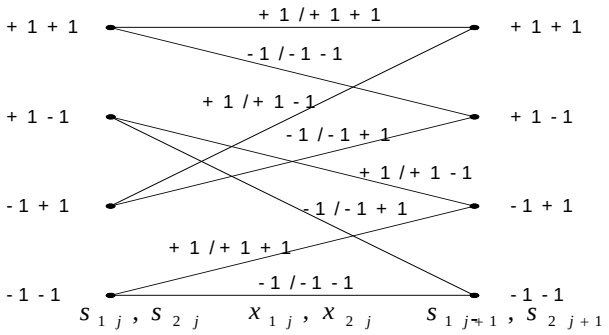


Fig. 4. Trellis of convolutional code

The encoded sequence $\underline{x} = (+1+1,+1+1,+1+1,+1+1,+1+1,+1+1,+1+1)$ (BPSK-modulated bits considered). In the received sequence there are some errors introduced. Fig.5. and Fig.6. show that hard-decision Viterbi algorithm is performing well in case of single errors.

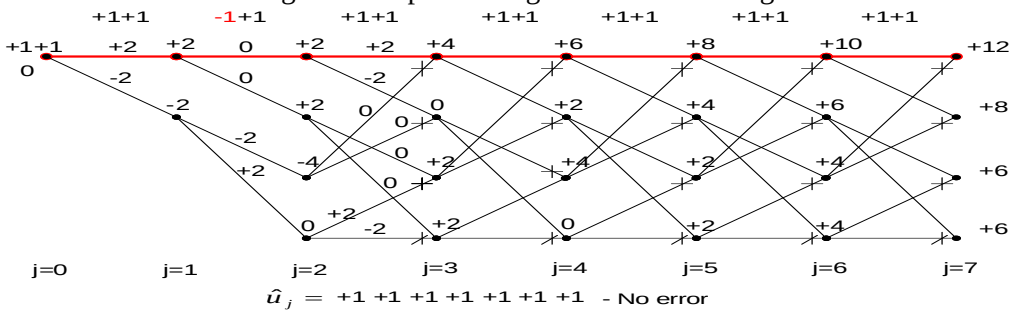


Fig. 5. Performance of Viterbi algorithm in case of one single error

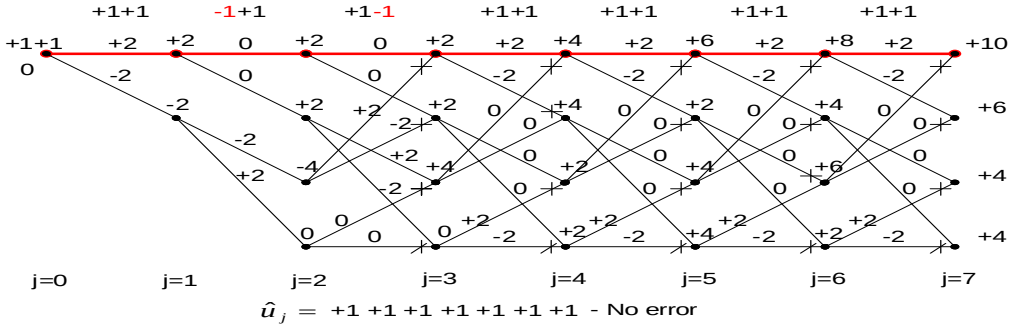


Fig. 6. Performance of Viterbi algorithm in case of two single errors

Fig.7. shows that the performance of hard-decision Viterbi algorithm degrades significantly in case of burst errors.

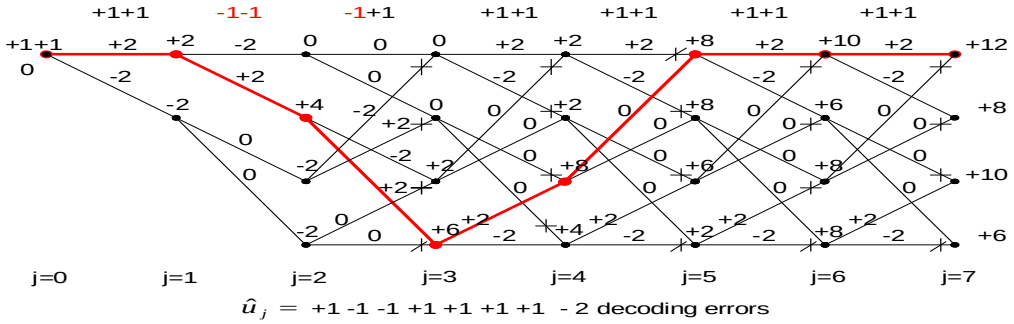


Fig. 7. Performance of the Viterbi algorithm in case of error bursts

Fig.8. shows that the application of the soft-decisions Viterbi algorithm improves the situation as the errors are corrected.

$$\lambda_j^{(m)} = x_{1j} l_{1j} y_{1j} + x_{2j} l_{2j} y_{2j}$$

l_j - channel state information $\quad l_j = \begin{cases} 2 & \text{“Good” channel} \\ 0.5 & \text{“Bad” channel} \end{cases}$

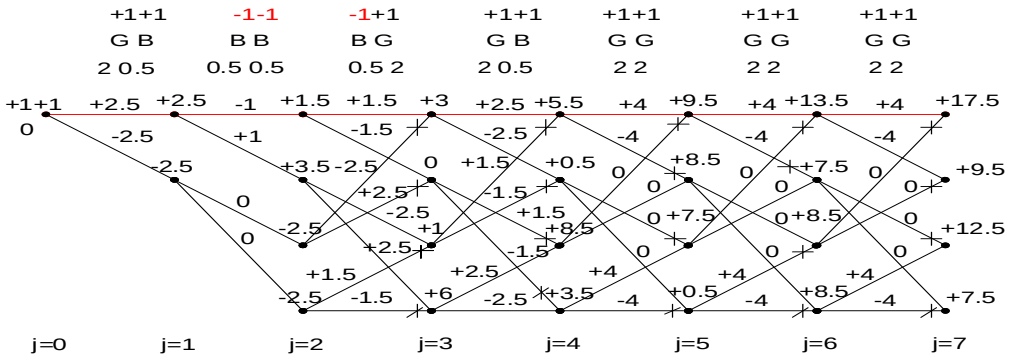


Fig. 8. Performance of the soft-decision Viterbi algorithm in case of error bursts

III. CONCLUSION

Convolutional codes are very easy to implement. Convolutional codes use smaller codewords in comparison to block codes, achieving the same quality. Puncturing techniques can be easily applied to convolutional codes. This allows generating a set of punctured codes out of one mother code. The advantage is that to decode them all only one decoder is needed and adaptive coding scheme can be thus implemented. One of the most important decoding algorithms is the Viterbi algorithm that uses the principle of maximum likelihood decoding. The Viterbi decoding uses hard decisions is therefore very vulnerable to error bursts. Using Soft instead of Hard decisions for Viterbi decoding improves the performance.

REFERENCES

[1] M. Tuechler and J. Hagenauer. “Channel coding” lecture script, Munich University of Technology, pp. 111-162, 2003