I. Convolutional Coding Encoder

The convolutional encoder is considered with $n = 2, K = 3, k = 1$. Therefore, each of the message bits is formed to double sequences. In other words, the impulse response of path 1 in figure 1, (1,1,1) can be denoted as $g_1(D) = 1 + D + D^2$ and the impulse response of path 2, (1,0,1) as $g_1(D) = 1 + D^2$. Consequently, the codeword is generated by $c_1(D) = g_1(D)m(D)$ and $c_2(D) = g_2(D)m(D)$. And then, the codeword is filled up sequentially; $\vec{c} = [c_{1,1}, c_{2,1}, c_{1,2}, c_{2,2}, \dots]$ such that $\overrightarrow{c_1} = [c_{1,1},\ c_{1,2}, \dots]$ and $\overrightarrow{c_2} = [c_{2,1},\ c_{2,2}, \dots]$. (ref. `function [m,c] = ConvolutionalEncoder(L)`)

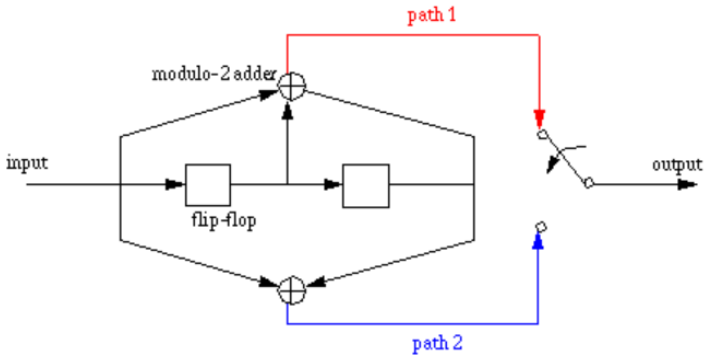

Figure 1. Convolutional encoder with n=2, K=3, k=1

II. Viterbi Decoding without sliding window

For the initial step, the left-most state of trellis as 0 goes to expectable branches on step 1 and 2 in figure 2. From the definition in the lecture note, there are precise definitions for each state such that $BM_k(s) = hamming\_distance(s, r_k)$ and $SM_k(s) = \sum BM_k(s)$ where $s \in \{00,01,10,11\}$. (ref. `function m_est = ViterbiDecoder(r))`

$$SM_2(00) = BM_2(00) + BM_1(00)$$

$$SM_2(01) = BM_2(10) + BM_1(11)$$

$$SM_2(10) = BM_2(11) + BM_1(00)$$

$$SM_2(11) = BM_2(01) + BM_1(11)$$

$$survivorPath(4,1:2) = \begin{bmatrix} 1 & 1 \\ 0 & 3 \\ 1 & 1 \\ 0 & 3 \end{bmatrix}$$

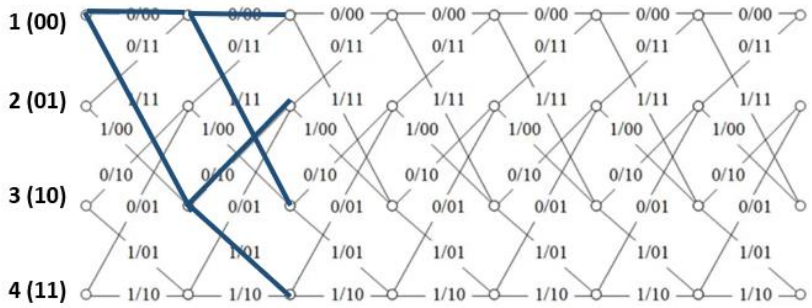

Figure 2. Initial 2 steps of trellis diagram

For the computation of k-th step, $k \geq 3$, we can simplify the process in figure 3 like that:

$$SM_k(00) = \min\{SM_{k-1}(00) + BM_k(00), SM_{k-1}(01) + BM_k(11)\}$$

$$SM_k(01) = \min\{SM_{k-1}(10) + BM_k(10), SM_{k-1}(11) + BM_k(01)\}$$

$$SM_k(10) = \min\{SM_{k-1}(00) + BM_k(11), SM_{k-1}(01) + BM_k(00)\}$$

$$SM_k(11) = \min\{SM_{k-1}(10) + BM_k(01), SM_{k-1}(11) + BM_k(10)\}$$

$$survivorPath(4,k) = \begin{bmatrix} idx\ (1\ or\ 2) \\ idx + 2\ (3\ or\ 4) \\ idx\ (1\ or\ 2) \\ idx + 2\ (3\ or\ 4) \end{bmatrix} \qquad where \quad SM_k(ind) = \min(SM_k)$$

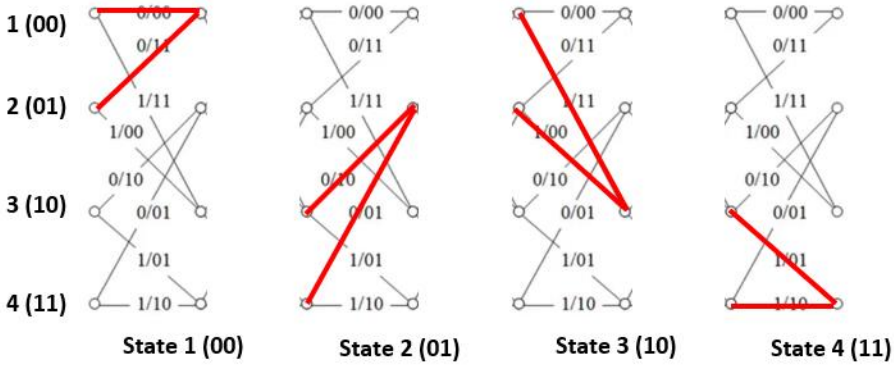

Figure 3. k-th step of trellis diagram, k>=3

In order to trace back the estimated message from the survivor path, the sequences are decoded by the state table in table 1. In addition, we start decoding from current state 00 since two 0s added at the end of the message bits during encoding. Finally, the two 0 bits in the end will be removed.

previous state / current state	00	01	10	11
00	0	0	·	·
01	·	·	0	0
10	1	1	·	·
11	·	·	1	1

Table 1. State table

As we can see the performance in figure 4, Viterbi decoding guarantees better BER above 4dB.

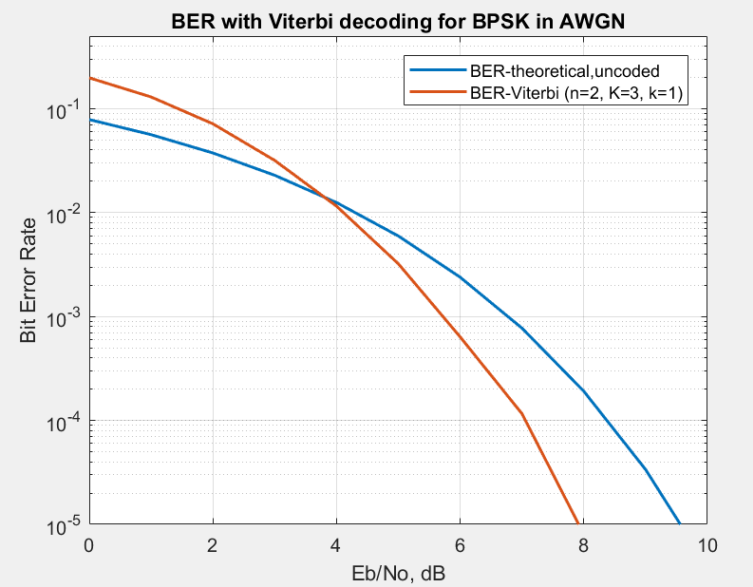

Figure 4. BER with a Viterbi decoder in L=10^6

III. Viterbi Decoding with sliding windows

By heuristic researches, the size of the sliding window, l, is reasonable on the order of 5 times the constraint length K of the convolutional code or more. Therefore, l should be equal to or more than 15, 5*3, in this example. Basically, each window has the same processes with Viterbi decoding without sliding windows but the size of the output (or survivorPath in the m file) is the first half as same as the size of the sliding window. As long as the block size is sufficient, forward processing calculates hamming distances and survivor paths with a double size of the window. Then, we assume that the first part of the windows is more reliable to detect original messages. The next processes, similarly, are starting from the final state from the previous decoding, and so on, in figure 4. (ref.

```
function m_est = Viterbi_window15(r,L))
```

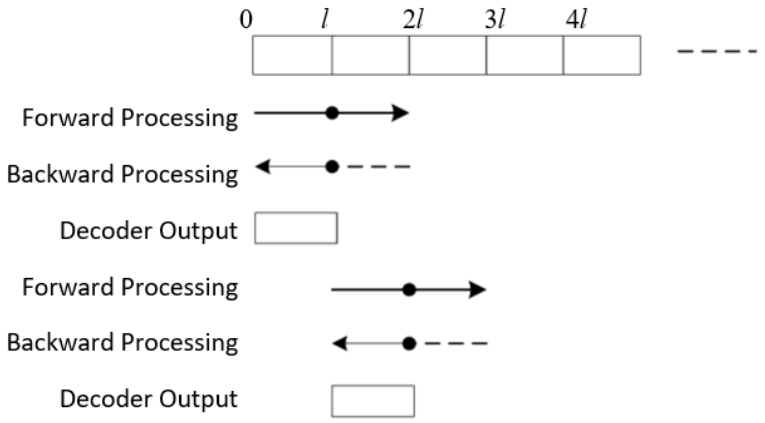

Figure 4. Windowing processes with trellis algorithm

The performance of the Viterbi decoder with the window size of 15 is better above 4dB in figure 5.

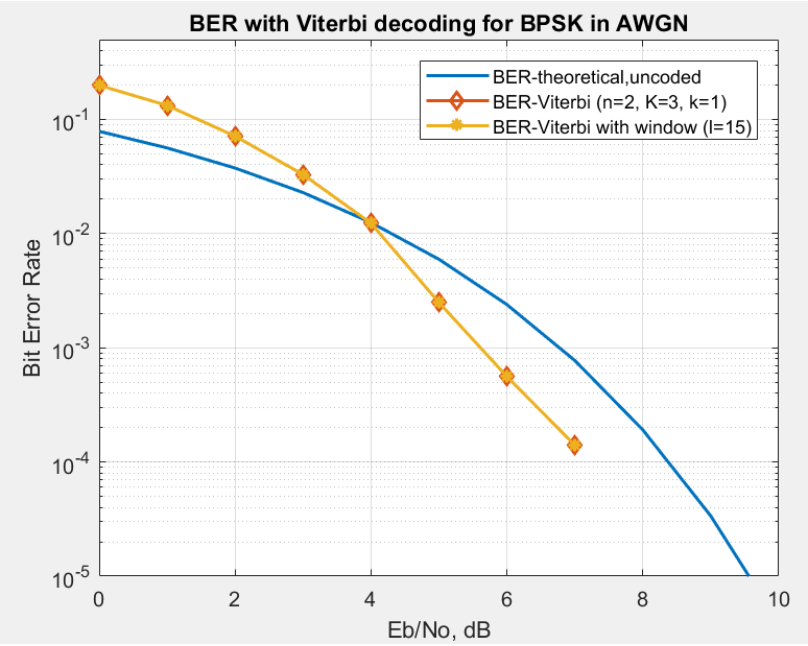

Figure 5. BER with a Viterbi decoder w/ and w/o the window size of 15 in L=10^5

Reference: Viterbi_window.m