Chapter 6 Turbo Codes

- 6.1 Introduction of Turbo Codes
- 6.2 Encoding of Turbo Codes
- 6.3 Decoding of Turbo Codes (Turbo Decoding)
- 6.4 Performance Analysis
§ 6.1 Introduction of Turbo Codes

- Invented by C. Berrou, A. Glavieux and P. Thitimajshima in 1993 [1].

- Integrate a couple of conv. codes in a parallel encoding structure. The two conv. codes are called the constituent codes of a turbo code.

- Exploit the interplay between the decoders of the two constituent codes in a soft information exchange decoding mechanism.

- Such a decoding mechanism is called turbo decoding, turbo decoding is NOT limited to decode turbo codes, but to any (serially or parallelly) concatenated code.

- Shannon capacity can be approached with the existence of error floor.

§ 6.1 Introduction of Turbo Codes

Why do we need code concatenation?
In BCJR decoding of a conv. code,

\[ \Gamma_{\Omega \rightarrow \Omega'} = P_a(u_{t'}) \cdot P_{ch}(c_{t'}^1) \cdot P_{ch}(c_{t'}^2) \]

channel observations

With a single conv. code, we do not have any knowledge of information bit \( u_{t'} \) and the \textit{a priori} prob. \( P_a(u_{t'} = 0) = P_a(u_{t'} = 1) = 0.5 \). With a couple of conv. codes that share the same information bits (but in different permutations), one decoder can gain the \textit{a priori} prob. of information bits \( u_{t'} \) from the output of the other decoder, and vice versa. As a result, BCJR decoding of each constituent code can be improved.
§ 6.1 Introduction of Turbo Codes

\begin{align*}
A \text{ priori prob.}: & \text{ knowledge about the information/coded bits before the decoding. It is also called the intrinsic prob.} \\
A \text{ posteriori prob.}: & \text{ knowledge about the information/coded bits after the decoding. It is used for estimation.} \\
\text{Extrinsic prob.}: & P_e = \frac{P_p}{P_a}, \text{ the extra knowledge (excluding the a priori prob.) delivered by the SISO decoder.}
\end{align*}
§ 6.2 Encoding of Turbo Codes

**Constituent codes:** Recursive Systematic Conv. (RSC) codes. Normally, the two constituent codes are the same.

**Interleaver (Π):** Generate a different information sequence (a permuted sequence) as the input to the RSC encoder (2). Normally, it is a random interleaver.

**Puncture:** Control the code rate.
§ 6.2 Encoding of Turbo Codes

- Given the binary message sequence as $\tilde{u} = [u_1, u_2, \ldots, u_k]$, output of the turbo encoder should be
  \[ \tilde{c} = [u_1 p_{t'}^{(1)} p_1^{(2)} u_2 p_{t'}^{(1)} p_2^{(2)} \ldots u_{t'} p_{t'}^{(1)} p_{t'}^{(2)} \ldots u_k p_k^{(1)} p_k^{(2)}]. \]

- Rate of the turbo code is $1/3$. To increase the rate to $1/2$, we can use puncturing whose pattern can be represented by
  \[ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]
  puncture $p_{t'}^{(2)}$ when $t'$ is odd \[ \uparrow \quad \uparrow \text{puncture } p_{t'}^{(1)} \text{ when } t' \text{ is even.} \]

- After puncturing, output of the turbo encoder should be
  \[ \tilde{c} = [u_1 p_{1}^{(1)} u_2 p_{2}^{(2)} \ldots u_k p_k^{(1)} (u_k p_k^{(2)})] \text{ when } k \text{ is odd} \quad \uparrow \quad \text{when } k \text{ is even.} \]
Example 6.1 Given the turbo encoder shown below with constituent code of the $(1, 1/5)_8$ conv. code. The puncturing pattern is $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. The interleaving pattern is $(8, 3, 7, 6, 9, 1, 10, 5, 2, 4)$. Determine the turbo codeword of message vector $\tilde{u} = [1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0]$. 
§ 6.2 Encoding of Turbo Codes

The original message vector
\[ \bar{u} = [1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0] \]
Output of the 1st constituent code is:
\[ \bar{p}^{(1)} = [1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0] \]
After interleaving, the permuted message vector becomes
\[ \bar{u}' = [0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1] \]
Output of the 2nd constituent code is:
\[ \bar{p}^{(2)} = [0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1] \]
Before puncturing, the turbo codeword is
\[ \bar{c} = [110 \ 000 \ 011 \ 111 \ 011 \ 100 \ 101 \ 000 \ 001 \ 001] \]
After puncturing, the turbo codeword is
\[ \bar{c} = [11 \ 00 \ 01 \ 11 \ 01 \ 10 \ 10 \ 00 \ 00 \ 01] \]
§ 6.3 Decoding of Turbo Codes

- **Parameterization**

  - Turbo codeword \( \bar{c} = [u_1 \ p_1^{(1)} \ p_1^{(2)}, u_2 \ p_2^{(1)} \ p_2^{(2)}, \ldots, u_k \ p_k^{(1)} \ p_k^{(2)}] \).
  
  - Assume the turbo codeword is transmitted using BPSK.
  
  - Received symbol vector
    \[
    \bar{y} = [y_1^{(0)} \ y_1^{(1)} \ y_1^{(2)}, y_2^{(0)} \ y_2^{(1)} \ y_2^{(2)}, \ldots, y_k^{(0)} \ y_k^{(1)} \ y_k^{(2)}].
    \]
  
  - Interleaved message vector
    \[
    \bar{u}' = \Pi(\bar{u}) = [u'_1, u'_2, \ldots, u'_k].
    \]
  
  - Interleaved (information) symbol vector
    \[
    \left[ y_1^{(0)'} \ y_2^{(0)'} \ldots \ y_k^{(0)'} \right] = \Pi([y_1^{(0)}, y_2^{(0)}, \ldots, y_k^{(0)}]).
    \]
§ 6.3 Decoding of Turbo Codes

Turbo decoding structure

- In BCJR (1), trellis transition probability is determined by
  \[ \Gamma_{\Omega \rightarrow \Omega'} = P_a(u_{t'}) P_{ch}(u_{t'}) P_{ch}(p_{t'}^{(1)}). \]

- In BCJR (2), trellis transition probability is determined by
  \[ \Gamma_{\Omega \rightarrow \Omega'} = P_a(u_{t'}') P_{ch}(u_{t'}') P_{ch}(p_{t'}^{(2)}). \]
§ 6.3 Decoding of Turbo Codes

Turbo decoding structure

- At the beginning of iterations, knowledge of information bits $u_t'$ is not available, and $P_a(u_t')$ are initialized as
  \[ P_a(u_t' = 0) = P_a(u_t' = 1) = 1/2. \]

- Once BCJR (1) delivers $P_e(u_t')$, knowledge of interleaved information bits $u_t'$ will be gained by mapping
  \[ \Pi(P_e(u_t')) \to P_a(u_t'), \]
  and BCJR (2) starts its decoding with $P_a(u_t')$, $P_{ch}(u_t')$, and $P_{ch}(p_t^{(2)})$. 
§ 6.3 Decoding of Turbo Codes

- Once BCJR (2) delivers $P_e(u_t')$, knowledge of information bits $u_t'$ will be gained by mapping
  \[ \Pi^{-1}(P_e(u_t')) \rightarrow P_a(u_t'), \]
  and BCJR (1) performs another round of decoding with $P_a(u_t')$, $P_{ch}(u_t')$ and $P_{ch}(p_t^{(1)})$.

- After a sufficient number of iterations, decisions will be made based on the *a posteriori* prob. $P_p(u_t')$ that are the deinterleaved version of output of BCJR (2), $P_p(u_t')$.

- If parity bits $p_t^{(1)}$ (or $p_t^{(2)}$) have been punctured, the channel observations become
  \[ P_{ch}(p_t^{(1)}) = 0 \] \[ P_{ch}(p_t^{(1)}) = 1 \] \[ P_{ch}(p_t^{(2)}) = 0 \] \[ P_{ch}(p_t^{(2)}) = 1 \]
  \[ 1/2 \] \[ 1/2 \]
  And all the channel observations remain unchanged during the whole iterative process.
6.3 Decoding of Turbo Codes

Advantage of systematic constituent codes

Using RSC

Encoding:

Transmission (coding rate is 1/3):

Decoding of the constituent codes:

\[ P_{\text{ch}}(u_t'), P_{\text{ch}}(p_t^{(1)}) \] are used in the 1st decoder, which is rate 1/2.

\[ P_{\text{ch}}(u_t'), P_{\text{ch}}(p_t^{(2)}) \] are used in the 2nd decoder, which is rate 1/2.
§ 6.3 Decoding of Turbo Codes

Using non-systematic constituent codes

Encoding:

Transmission (coding rate is 1/4):

Decoding of the constituent codes:

\[ P_{ch}\left(c_{2t' - 1}^{(1)}\right), P_{ch}\left(c_{2t'}^{(1)}\right) \] are used in the 1st decoder, which is rate 1/2.

\[ P_{ch}\left(c_{2t' - 1}^{(2)}\right), P_{ch}\left(c_{2t'}^{(2)}\right) \] are used in the 2nd decoder, which is rate 1/2.
§ 6.3 Decoding of Turbo Codes

Using non-systematic constituent codes

\[
\begin{align*}
&\text{Enc. (1)} \rightarrow c_{2t'}^{(1)} c_{2t'}^{(1)} \\
&\text{Enc. (2)} \rightarrow c_{2t'-1}^{(2)} c_{2t'}^{(2)}
\end{align*}
\]

\( u_t' \)

In realizing a rate 1/3 coded transmission, we may puncture one coded bit in each time instant. E.g., (i) puncture \( c_{2t'}^{(2)} \) as

\[
\begin{align*}
c_1^{(1)} & \quad c_2^{(1)} & \quad c_2^{(2)} & \quad c_3^{(1)} & \quad c_4^{(1)} & \quad c_3^{(2)} & \quad \ldots & \quad c_{2t'-1}^{(1)} & \quad c_{2t'}^{(1)} & \quad c_{2t'}^{(2)} & \quad \ldots
\end{align*}
\]

or (ii) puncture \( c_{2t'-1}^{(2)} \) when \( t' \) is odd, and \( c_{2t'-1}^{(1)} \) when \( t' \) is even, as

\[
\begin{align*}
c_1^{(1)} & \quad c_2^{(1)} & \quad c_2^{(2)} & \quad c_4^{(1)} & \quad c_3^{(1)} & \quad c_4^{(2)} & \quad \ldots & \quad c_{2t'-1}^{(1)} & \quad c_{2t'}^{(1)} & \quad c_{2t'}^{(2)} & \quad (c_{2t'}^{(1)} c_{2t'-1}^{(2)} c_{2t'}^{(2)}) & \quad \ldots
\end{align*}
\]

Decoding of the constituent codes:

In case (i)

\( P_{\text{ch}} \left( c_{2t'-1}^{(1)} \right), P_{\text{ch}} \left( c_{2t'}^{(1)} \right) \) are used in the 1st decoder, which is rate 1/2.

\( P_{\text{ch}} \left( c_{2t'-1}^{(2)} \right) \) are used in the 2nd decoder, which is forced to be rate 1.

In case (ii)

The 1st and 2nd decoders are forced to be rate 2/3.
### § 6.3 Decoding of Turbo Codes

**Example 6.2**  Message vector \( \bar{u} = [1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0] \)

Transmitted codeword \( \bar{c} = [11 \ 00 \ 01 \ 11 \ 01 \ 10 \ 10 \ 00 \ 00 \ 01] \)

Received symbol \( \bar{y} = [1.66, \ 2.49, \ -2.35, \ -1.39, \ 0.22, \ 1.27, \ -0.41, \ 0.30, \ -2.00, \ 1.16, \ 1.70, \ -1.69, \ 0.90, \ -0.38, \ -3.28, \ -0.82, \ 0.12, \ -1.30, \ -3.31, \ 2.28] \).

After iteration 1:

<table>
<thead>
<tr>
<th>( P_e(u_{t'} = 0) )</th>
<th>0.01</th>
<th>0.32</th>
<th>0.99</th>
<th>0.04</th>
<th>0.84</th>
<th>0.69</th>
<th>0.37</th>
<th>0.32</th>
<th>0.92</th>
<th>0.32</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_e(u_{t'} = 1) )</td>
<td>0.99</td>
<td>0.68</td>
<td>0.01</td>
<td>0.96</td>
<td>0.16</td>
<td>0.31</td>
<td>0.63</td>
<td>0.68</td>
<td>0.08</td>
<td>0.68</td>
</tr>
<tr>
<td>( P_e(u_{t'} = 0) )</td>
<td>0.50</td>
<td>0.82</td>
<td>0.50</td>
<td>0.08</td>
<td>0.50</td>
<td>0.67</td>
<td>0.50</td>
<td>0.23</td>
<td>0.50</td>
<td>0.03</td>
</tr>
<tr>
<td>( P_e(u_{t'} = 1) )</td>
<td>0.50</td>
<td>0.18</td>
<td>0.50</td>
<td>0.92</td>
<td>0.50</td>
<td>0.33</td>
<td>0.50</td>
<td>0.77</td>
<td>0.50</td>
<td>0.97</td>
</tr>
<tr>
<td>( P_p(u_{t'} = 0) )</td>
<td>0.00</td>
<td>0.27</td>
<td>1.00</td>
<td>0.49</td>
<td>1.00</td>
<td>0.31</td>
<td>0.28</td>
<td>0.02</td>
<td>1.00</td>
<td>0.07</td>
</tr>
<tr>
<td>( P_p(u_{t'} = 1) )</td>
<td>1.00</td>
<td>0.73</td>
<td>0.00</td>
<td>0.51</td>
<td>0.00</td>
<td>0.69</td>
<td>0.72</td>
<td>0.98</td>
<td>0.00</td>
<td>0.93</td>
</tr>
</tbody>
</table>

**Note:** Only the real part of the received symbols are preserved here.
§ 6.3 Decoding of Turbo Codes

After iteration 2:

<table>
<thead>
<tr>
<th>$P_e(u_t' = 0)$</th>
<th>0.00</th>
<th>0.93</th>
<th>0.99</th>
<th>0.01</th>
<th>0.91</th>
<th>0.07</th>
<th>0.37</th>
<th>0.93</th>
<th>0.93</th>
<th>0.93</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_e(u_t' = 1)$</td>
<td>1.00</td>
<td>0.07</td>
<td>0.01</td>
<td>0.99</td>
<td>0.09</td>
<td>0.93</td>
<td>0.63</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
</tr>
<tr>
<td>$P_e(u_t' = 0)$</td>
<td>0.50</td>
<td>0.99</td>
<td>0.50</td>
<td>0.04</td>
<td>0.50</td>
<td>0.14</td>
<td>0.50</td>
<td>0.34</td>
<td>0.50</td>
<td>0.01</td>
</tr>
<tr>
<td>$P_e(u_t' = 1)$</td>
<td>0.50</td>
<td>0.01</td>
<td>0.50</td>
<td>0.96</td>
<td>0.50</td>
<td>0.86</td>
<td>0.50</td>
<td>0.66</td>
<td>0.50</td>
<td>0.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P_p(u_t' = 0)$</th>
<th>0.00</th>
<th>0.92</th>
<th>1.00</th>
<th>0.11</th>
<th>1.00</th>
<th>0.01</th>
<th>0.28</th>
<th>0.32</th>
<th>1.00</th>
<th>0.68</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_p(u_t' = 1)$</td>
<td>1.00</td>
<td>0.08</td>
<td>0.00</td>
<td>0.89</td>
<td>0.00</td>
<td>0.99</td>
<td>0.72</td>
<td>0.68</td>
<td>0.00</td>
<td>0.32</td>
</tr>
</tbody>
</table>

After iteration 3:

<table>
<thead>
<tr>
<th>$P_e(u_t' = 0)$</th>
<th>0.00</th>
<th>0.97</th>
<th>0.99</th>
<th>0.01</th>
<th>0.94</th>
<th>0.04</th>
<th>0.37</th>
<th>0.96</th>
<th>0.93</th>
<th>0.96</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_e(u_t' = 1)$</td>
<td>1.00</td>
<td>0.03</td>
<td>0.01</td>
<td>0.99</td>
<td>0.06</td>
<td>0.96</td>
<td>0.63</td>
<td>0.04</td>
<td>0.07</td>
<td>0.04</td>
</tr>
<tr>
<td>$P_e(u_t' = 0)$</td>
<td>0.50</td>
<td>0.99</td>
<td>0.50</td>
<td>0.03</td>
<td>0.50</td>
<td>0.09</td>
<td>0.50</td>
<td>0.37</td>
<td>0.50</td>
<td>0.01</td>
</tr>
<tr>
<td>$P_e(u_t' = 1)$</td>
<td>0.50</td>
<td>0.01</td>
<td>0.50</td>
<td>0.97</td>
<td>0.50</td>
<td>0.91</td>
<td>0.50</td>
<td>0.63</td>
<td>0.50</td>
<td>0.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P_p(u_t' = 0)$</th>
<th>0.00</th>
<th>0.96</th>
<th>1.00</th>
<th>0.10</th>
<th>1.00</th>
<th>0.00</th>
<th>0.28</th>
<th>0.49</th>
<th>1.00</th>
<th>0.82</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_p(u_t' = 1)$</td>
<td>1.00</td>
<td>0.04</td>
<td>0.00</td>
<td>0.90</td>
<td>0.00</td>
<td>1.00</td>
<td>0.72</td>
<td>0.51</td>
<td>0.00</td>
<td>0.18</td>
</tr>
</tbody>
</table>
§ 6.3 Decoding of Turbo Codes

After iteration 4:

<table>
<thead>
<tr>
<th></th>
<th>$P_e(u'_t = 0)$</th>
<th>$P_e(u'_t = 1)$</th>
<th>$P_e(u'_t = 0)$</th>
<th>$P_e(u'_t = 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_e(u'_t = 0)$</td>
<td>0.00 0.97 0.99</td>
<td>0.01 0.94 0.04</td>
<td>0.37 0.97 0.93</td>
<td>0.97 0.93 0.97</td>
</tr>
<tr>
<td>$P_e(u'_t = 1)$</td>
<td>1.00 0.03 0.01</td>
<td>0.99 0.06 0.96</td>
<td>0.63 0.03 0.07</td>
<td>0.03 0.07 0.03</td>
</tr>
<tr>
<td>$P_e(u'_t = 0)$</td>
<td>0.50 0.99 0.50</td>
<td>0.03 0.50 0.09</td>
<td>0.50 0.37 0.50</td>
<td>0.01 0.50 0.01</td>
</tr>
<tr>
<td>$P_e(u'_t = 1)$</td>
<td>0.50 0.01 0.50</td>
<td>0.97 0.50 0.91</td>
<td>0.50 0.63 0.50</td>
<td>0.50 0.99 0.50</td>
</tr>
<tr>
<td>$P_p(u'_t = 0)$</td>
<td>0.00 0.97 1.00</td>
<td>0.10 1.00 0.00</td>
<td>0.28 0.51 1.00</td>
<td>0.82 1.00 0.82</td>
</tr>
<tr>
<td>$P_p(u'_t = 1)$</td>
<td>1.00 0.03 0.00</td>
<td>0.90 0.00 1.00</td>
<td>0.72 0.49 0.00</td>
<td>0.18 0.49 0.18</td>
</tr>
</tbody>
</table>

**Observations:**

(i) The iterative decoding corrects 3 errors;
(ii) Let $L_p(u'_t) = \ln \frac{P_p(u'_t = 0)}{P_p(u'_t = 1)}$. If $|L_p(u'_t)|$ is greater, the decision made on $u'_t$ will be more confident;
(iii) The decisions made on the message bits become more confident as the iteration progresses.
§ 6.3 Decoding of Turbo Codes

Log-MAP BCJR
Define function
\[ \max^*(x, y) \triangleq \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|}) \]
whose multivariate version can be defined recursively as
\[ \max^*(x, y, z) \triangleq \ln(e^x + e^y + e^z) = \max^*(\max^*(x, y), z) \]
Similarly, let \( X \) be a finite set so that
\[ \max^*(X) \triangleq \ln \sum_{x \in X} e^x \]
Define BCJR metrics in log domain
\[ \Gamma_{\Omega \rightarrow \Omega'}^* = \ln \Gamma_{\Omega \rightarrow \Omega'} \]
\[ A_{t'}^*(\Omega) = \ln A_{t'}(\Omega) = \ln \sum_{(\Omega_0, \Omega)} e^{\left[A_{t'-1}^*(\Omega_0) + \Gamma_{\Omega_0 \rightarrow \Omega}^*\right]} \]
\[ = \max^*(\{A_{t'-1}^*(\Omega_0) + \Gamma_{\Omega_0 \rightarrow \Omega}^* | \forall (\Omega_0, \Omega)\}) \]
§ 6.3 Decoding of Turbo Codes

Similarly, \( B^*_t + 1(\Omega') = \ln B^*_t + 1(\Omega') = \ln \sum_{(\Omega', \Omega'')} e^{B^*_t + 2(\Omega'')} + \Gamma^*_{\Omega' \rightarrow \Omega''} \)

\[
= \max^*\{B^*_t + 2(\Omega'') + \Gamma^*_{\Omega' \rightarrow \Omega''} | \forall (\Omega', \Omega'')\}
\]

\[
L(u_t') = \max^*_{(\Omega \rightarrow \Omega')_0} \left( A^*_t(\Omega) + \Gamma^*_{\Omega \rightarrow \Omega'} + B^*_t + 1(\Omega') \right) - \max^*_{(\Omega \rightarrow \Omega')_1} \left( A^*_t(\Omega) + \Gamma^*_{\Omega \rightarrow \Omega'} + B^*_t + 1(\Omega') \right)
\]

The max-log-MAP algorithm is to replace \( \max^*(\cdot) \) by \( \max(\cdot) \).

**Remark:** Turbo decoding efficiency can be improved by the so-called log-MAP algorithm [1] or the max-log-MAP algorithm [2]. Both of the algorithms deal with log-likelihood ratios rather than probabilities. The max-log-MAP algorithm has a computational complexity of not more than three times of Viterbi algorithm, but suffers a slight performance loss compared to BCJR and log-MAP algorithms.


§ 6.4 Performance Analysis

BER performance of rate half turbo code with constituent code of (1, 1/5) RSC over AWGN channel using BPSK.
§ 6.4 Performance Analysis

The BER performance of the classical turbo code with length 1024 and rate 1/3 [1].

§ 6.4 Performance Analysis

**Q: Why there is an error floor?**

- The bit error rate (BER) (denoted as \( P_b \)) of a conv. code (and turbo code) is determined by

\[
P_b \leq \sum_{i=1}^{2^k} \frac{w_i}{k} Q\left(\sqrt{\frac{2d_i \cdot R \cdot E_b}{N_0}}\right).
\]

- Let \( \bar{u}_i \) denote a message vector and \( \bar{c}_i \) denote its corresponding codeword, \( w_i = \text{weight}(\bar{u}_i) \) and \( d_i = \text{weight}(\bar{c}_i) \).

- \( k = \text{length}(\bar{u}_i) \) and there are \( 2^k \) codewords in the codebook.

- \( R \) is the rate of the code.

- \( \frac{E_b}{N_0} \) — signal-to-noise ratio (SNR).

\[
Q \text{ function as } Q(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-\frac{v^2}{2}} dv.
\]
§ 6.4 Performance Analysis

- Since \( d_i = d_{\text{free}}, d_{\text{free}} + 1, \ldots, \frac{k}{R} \), by grouping terms with the same \( d_i \), the above inequality can be written as:

\[
P_b \leq \sum_{d=d_{\text{free}}}^{\frac{k}{R}} \frac{W_d}{k} Q\left(\sqrt{\frac{2d \cdot R \cdot E_b}{N_0}}\right)
= \sum_{d=d_{\text{free}}}^{\frac{k}{R}} \frac{\tilde{w}_d N_d}{k} Q\left(\sqrt{\frac{2d \cdot R \cdot E_b}{N_0}}\right).
\]

- \( \tilde{w}_d \) — weight of message vectors that correspond to codeword of weight \( d \).
- \( N_d \) — Number of codewords of weight \( d \).
- \( W_d \) — Total weight of message vectors that correspond to codeword of weight \( d \).
§ 6.4 Performance Analysis

- When the SNR \( \left( \frac{E_b}{N_0} \right) \) increases, the asymptotic behavior of \( P_b \) is dominated by the first term in the summation as

\[
P_b \approx \frac{N_{d_{\text{free}}}}{k} \hat{w}_{d_{\text{free}}} Q \left( \sqrt{\frac{2d_{\text{free}} \cdot R \cdot E_b}{N_0}} \right).
\]

- In the log \( P_b \) vs. log \( \frac{E_b}{N_0} \) graph, \( d_{\text{free}} \) determines the slope of the BER vs. SNR (dB) curve.

Remark: The error floor at high SNR is due to a small \( d_{\text{free}} \), or alternatively the presence of low weight codewords.
§ 6.4 Performance Analysis

Motivation of having an interleaver between the two encoders: Try to avoid the low weight conv. codewords and subsequently the low weight turbo codeword being produced.

**Example 6.3** Following the encoder structure of *Example 6.1*, if the message vector \( \tilde{u} = [0 0 0 0 1] \), the output of the RSC (1) will be

\[
\tilde{c}_1 = [00 00 00 00 11].
\]

Without interleaving, the output of RSC (2) will be the same as RSC (1) as \( \tilde{c}_2 = \tilde{c}_1 \). And the turbo codeword is

\[
\tilde{c} = [000 000 000 000 111].
\]

With interleaving, \( \tilde{u}' = [1 0 0 0 0] \), the output of RSC (2) will now be

\[
\tilde{c}_2 = [11 00 01 00 01].
\]

And the turbo codeword becomes

\[
\tilde{c} = [001 000 001 000 111].
\]