Spreading Codes and Characteristics
and
Error Correction Codes

Global Navigational Satellite Systems (GNSS-16)
Short course, NERTU

Prasad Krishnan

International Institute of Information Technology, Hyderabad

December, 2016
Spreading Codes
Talk Outline - Spreading Codes

- Spreading codes idea
- Need for spreading codes in GNSS
- Generating spreading codes for GNSS
What are Spreading Codes?

Example

- Consider that you have a message bit $b$ (can be zero or one).
- Instead of transmitting $b$ we transmit $-b[1\ 0\ 0\ 0\ 0]$.
- The vector $[1\ 0\ 0\ 0\ 0]$ is like a carrier - we call it the code.
- To decode, multiply received vector by $[1\ 0\ 0\ 0\ 0]^T$. 
What are Spreading Codes?

Example

- 4 different bits with 4 orthogonal codes transmitted at the same time.

\[
b_0[1 0 0 0] + b_1[0 1 0 0] + b_2[0 0 1 0] + b_3[0 0 0 1] = b_0 x_0 + b_1 x_1 + b_2 x_2 + b_3 x_3
\]
What are Spreading Codes?

Example

- 4 different bits with 4 **orthogonal** codes transmitted at the same time.

\[
b_0 [1 \ 0 \ 0 \ 0] + b_1 [0 \ 1 \ 0 \ 0] + b_2 [0 \ 0 \ 1 \ 0] + b_3 [0 \ 0 \ 0 \ 1] = b_0 x_0 + b_1 x_1 + b_2 x_2 + b_3 x_3
\]

- Note that \(x_i x_j^T = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}\)

- Can get the bit \(b_i\) by multiplying with \(x_i^T\). (\(i\) represents the shift in time).
Example

- Consider that a coded bit $b$ comes with an arbitrary unknown delay ($0 \leq j \leq 3$).
- Received vector $= bx_j$ ($0 \leq j \leq 3$).
- Then can we find out the delay $j$ and bit $b$?
- Multiplying with $x_i$ ($0 \leq i \leq 3$) does the trick.
More generally - Pseudo-Random-Noise Sequences

- Consider two or more binary sequences (\(\{+1, -1\}\)) sequence which have 'good' autocorrelation properties and 'good' cross correlation properties.
- Such sequences are called Pseudo-Random-Noise sequences.
More generally - Pseudo-Random-Noise Sequences

- Consider two or more binary sequences \( \{+1, -1\} \) sequence which have 'good' autocorrelation properties and 'good' cross correlation properties.
- Such sequences are called Pseudo-Random-Noise sequences.

'Good' autocorrelation property of a sequence \( x \)

- \( x_i x_j^T \) has a high value if \( i = j \) and low value if \( i \neq j \).

'Good' cross-correlation property of sequences \( x \) and \( y \)

- \( x_i y_j^T \) has a low value for any \( i, j \).
How are PNR sequences useful in GNSS?

Each satellite has its own unique PRN sequence, and uses it to modulate data transmitted to receivers.

Ranging

- Good autocorrelation properties → Find Delay due to separation between Rx and Satellite Tx.
- Delay (from multiple satellites) → User Location.

Saving spectrum

- Good cross-correlation properties → Decode info from different satellites.
- Multiple satellites can transmit over the same frequency.

Gain in SNR

- Same bit is encoded in a long PRN sequence → redundancy.
- Redundancy → provides SNR gain (and hence lowers prob. of error).
How are PNR sequences useful in GNSS?

Each satellite has its own unique PRN sequence, and uses it to modulate data transmitted to receivers.

**Ranging**

- Good autocorrelation properties → Find Delay due to separation between Rx and Satellite Tx.
- Delay (from multiple satellites) → User Location.

**Saving spectrum**

- Good cross-correlation properties → Decode info from different satellites.
- Multiple satellites can transmit over the same frequency.
How are PNR sequences useful in GNSS?

Each satellite has its own unique PRN sequence, and uses it to modulate data transmitted to receivers.

**Ranging**

- Good autocorrelation properties → Find Delay due to separation between Rx and Satellite Tx.
- Delay (from multiple satellites) → User Location.

**Saving spectrum**

- Good cross-correlation properties → Decode info from different satellites.
- Multiple satellites can transmit over the same frequency.

**Gain in SNR**

- Same bit is encoded in a long PRN sequence → redundancy.
- Redundancy → provides SNR gain (and hence lowers prob. of error).
Generating 'Pseudo-Random-Noise'

- PN Sequences deterministically generated, yet possess properties of randomly generated sequences

- PN sequences generated using *linear feedback shift registers*. 
Linear Feedback Shift Registers

- M-sequences (binary maximal length shift-register sequence) are generated using linear feedback shift-register and exclusive OR-gate circuits.

- Linear generator polynomial $g(x)$ of degree $m > 0$

  $$g(x) = x^m + x^{m-1} + x^{m-2} + \ldots + g_1 x + g_0$$

- Recurrence Equation (1)

  $$c_i + m = g_{m-1} c_{i+m-1} + g_{m-2} c_{i+m-2} + \ldots + g_1 c_{i+1} + c_i \pmod{2}$$

- Output sequence is given by

  $$c_{i+m} = g_{m-1} c_{i+m-1} + g_{m-2} c_{i+m-2} + \ldots + g_1 c_{i+1} + c_i \pmod{2}$$
Characteristic Polynomial of LFSR

- Since the operation is binary addition, the above output equation can be rewritten as

\[ \sum_{\ell=0}^{m} g_{\ell} c_{i+\ell} = 0, \]

where \( g_0 = g_m = 1 \)

- The characteristic polynomial of the LFSR is given by

\[ g(x) = \sum_{\ell=0}^{m} g_{\ell} x^{\ell} \]

- We are interested in special kinds of LFSRs which can output PRN sequences.
Every polynomial \( g(x) \) with coefficients in binary field having \( g(0) = 1 \) divides \( x^N + 1 \) for some \( N \). The smallest \( N \) for which this is true is called the period of \( g(x) \).

An irreducible polynomial of degree \( m \) whose period is \( 2^m - 1 \) is called a primitive polynomial.
m-Sequences: Examples of PRN sequences

- An LFSR produces an *m*-sequence (maximum length) if and only if its characteristic polynomial is a primitive polynomial.
- In the above example, the polynomial is \( g(x) = 1 + x^3 + x^{10} \)
Delay and Add Property of m-Sequences

- The cyclic shift of an $m$-sequence is also an $m$-sequence
- The sum of an $m$-sequence and a cyclic shift of itself is also an $m$-sequence
Let \((s_t)\) be an \(m\)-sequence of period \(N = 2^n - 1\). Then the autocorrelation of the \(m\)-sequence is

\[
\theta_{s,s}(\tau) = \begin{cases} 
2^n - 1 & \text{if } \tau = 0 \left( \text{mod} 2^n - 1 \right) \\
-1 & \text{if } \tau \neq 0 \left( \text{mod} 2^n - 1 \right)
\end{cases}
\]
Property of m-sequences (2)

Property VI – The Shift and Add Property

The sum of an m-sequence and a cyclic shift of itself (mod 2, term by term) is another m-sequence.

Property VII – Thumb-Tack Autocorrelation

The normalized periodic autocorrelation function of an m-sequence, defined as:

$$
\rho(\tau) = \sum_{n=0}^{N-1} c(n) c(n+\tau) 
$$

is equal to 1 (mod N) and $-1/N$ for $0 \neq i (\mod N)$.

- Proven easily by shift and add property.
Gold Sequences

- m-Sequences have good auto-correlation properties but poor cross-correlation properties (cross-correlation can be high, which we don’t want).

- Two m-Sequence generators are used to generate a “Gold Sequence” which has good cross-correlation properties.
Generating Gold Sequences

10.23 MHz clock
\[\div 10\]

G1 generator
\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

G1 code

G2 generator
\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

G2 code

C/A code (Gold code)

Phase selector
\[
\begin{array}{cccccccccc}
S1 & S2 \\
\end{array}
\]

G1 code

C/A code

Reset circuit

Epoch detector
\[\div 20\]

50Hz clock for navigation data
Gold Sequences in GPS

<table>
<thead>
<tr>
<th>Satellite 1D Number</th>
<th>GPS PRN Number</th>
<th>Code Phase Selection (G2)</th>
<th>Code Delay Chips</th>
<th>First 10 Chips Octal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2 eight 6</td>
<td>5</td>
<td>1440</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3 eight 7</td>
<td>6</td>
<td>1620</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4 eight 8</td>
<td>7</td>
<td>1710</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5 eight 9</td>
<td>8</td>
<td>1744</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1 one 9</td>
<td>17</td>
<td>1133</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2 one 10</td>
<td>18</td>
<td>1455</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1 one 8</td>
<td>139</td>
<td>1131</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2 one 9</td>
<td>140</td>
<td>1454</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>3 one 10</td>
<td>141</td>
<td>1626</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>2 one 3</td>
<td>251</td>
<td>1504</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>3 one 4</td>
<td>252</td>
<td>1642</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>5 one 6</td>
<td>254</td>
<td>1750</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>6 one 17</td>
<td>255</td>
<td>1764</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>7 one 8</td>
<td>256</td>
<td>1772</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>8 one 9</td>
<td>257</td>
<td>1775</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>9 one 10</td>
<td>258</td>
<td>1776</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>1 one 4</td>
<td>469</td>
<td>1156</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>2 one 5</td>
<td>470</td>
<td>1467</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>3 one 6</td>
<td>471</td>
<td>1633</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>4 one 7</td>
<td>472</td>
<td>1715</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>5 one 8</td>
<td>473</td>
<td>1746</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>6 one 9</td>
<td>474</td>
<td>1763</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>1 one 3</td>
<td>509</td>
<td>1063</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>4 one 6</td>
<td>512</td>
<td>1706</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>5 one 7</td>
<td>513</td>
<td>1743</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>6 one 8</td>
<td>514</td>
<td>1761</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>7 one 9</td>
<td>515</td>
<td>1770</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>8 one 10</td>
<td>516</td>
<td>1774</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>1 one 6</td>
<td>859</td>
<td>1127</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>2 one 7</td>
<td>860</td>
<td>1453</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>3 one 8</td>
<td>861</td>
<td>1625</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>4 one 9</td>
<td>862</td>
<td>1712</td>
</tr>
<tr>
<td>-</td>
<td>33</td>
<td>5 one 10</td>
<td>863</td>
<td>1745</td>
</tr>
<tr>
<td>-</td>
<td>34</td>
<td>4 one 10</td>
<td>950</td>
<td>1713</td>
</tr>
<tr>
<td>-</td>
<td>35</td>
<td>1 one 7</td>
<td>947</td>
<td>1134</td>
</tr>
<tr>
<td>-</td>
<td>36</td>
<td>2 one 8</td>
<td>948</td>
<td>1456</td>
</tr>
<tr>
<td>-</td>
<td>37</td>
<td>4 one 10</td>
<td>950</td>
<td>1713</td>
</tr>
</tbody>
</table>
Forward Error Correction
Talk Outline - Error Correcting Codes

- Need for Error Correction
- Error Correcting Codes (general principles and examples)
- Encoding and Decoding of a Block Code (Hamming Code)
- Types of Error Correcting Codes (in GNSS)
- Encoding and Decoding of Convolutional Codes
Channel Coding

- The bit cross-over probability $p$ ($< 0.5$) is a property of the channel.
- Free to manipulate the input and output to the channel.
- Encode messages to *codewords* (add redundancy cleverly): Reduce effective $\text{Prob}(error)$. 

\[ p : \text{cross-over probability, say 0.1} \]
A trivial code example - Repetition code

- Repeat the same bit three times
- Message 0 $\rightarrow [0 \ 0 \ 0]$ (codeword), Message 1 $\rightarrow [1 \ 1 \ 1]$.  
- Decode by Majority logic.
- For the above channel, probability of error comes down (Check!).
General ideas behind FEC for the binary symmetric channel

- Decoder decides the Tx codeword $c$ based on received Noisy Codeword $y$.
- Decoding rule: Decoding for the most likely codeword. (Choose that $c$ which maximizes $p(y|c)$).
General ideas behind FEC for the binary symmetric channel

- Decoder decides the Tx codeword $c$ based on received Noisy Codeword $y$.
- Decoding rule: Decoding for the **most likely codeword**. (Choose that $c$ which maximizes $p(y|c)$).
- Probability that every transmitted bit is flipped is $p < 0.5$
- If you don’t code at all, the rule decodes to the bit that was received as it is.
For a code of length $n$: Choose the codeword which is closest to Rx Vector $y$ in terms of number of flipped bits.

Minimum Hamming Distance Rule.
General ideas behind FEC for the binary symmetric channel

- For a code of length $n$: Choose the codeword which is closest to Rx Vector $y$ in terms of number of flipped bits.
- Minimum Hamming Distance Rule.

Example

Repetition Code

- The Majority Decoder is in fact the Minimum Hamming Distance Decoder.
- The Repetition Code can correct any single-bit error.
For a code of length $n$: Choose the codeword which is closest to Rx Vector $\mathbf{y}$ in terms of number of flipped bits.

Minimum Hamming Distance Rule.

Example

Repetition Code

- The Majority Decoder is in fact the Minimum Hamming Distance Decoder.
- The Repetition Code can correct any single-bit error.
- Increase $n$ to decrease $P(\text{error})$.
- Rate $\frac{1}{n}$: For 1 bit of message, we need to send $n$ coded bits. (Pretty bad!)
Can we do better? - Linear Block Codes

- Each set of \( k \) message bits maps to a unique codeword
- Each of the \( n \) bits is a linear combination of \( k \) message bits
- \( n = \) length of the code, \( k = \) dimension of the code
Linear Block Codes

- Let \( \mathbf{u} \) be the message vector and \( \mathbf{c} \) is the corresponding codeword.
- How to get \( \mathbf{c} \) from \( \mathbf{u} \)?
- Linear Block Codes used a Linear Map \( \mathbf{c} = \mathbf{u} \mathbf{G} \)

- \( \mathbf{G} \) is a full-rank matrix of size \( k \times n \) \( (k \leq n) \). The code is a \((n, k)\) linear block code.
Linear Block Codes

- Let $u$ be the message vector and $c$ is the corresponding codeword.
- How to get $c$ from $u$?
- Linear Block Codes used a Linear Map

$$c = uG$$

- $G$ is a full-rank matrix of size $k \times n$ $(k \leq n)$. The code is a $(n, k)$ linear block code.

Repetition Code

- $G = (1 \ 1 \ 1)$.
- Message $u \in \{1, 0\}$.
- Codewords are $[1 \ 1 \ 1]$ and $[0 \ 0 \ 0]$. 
Linear Block Codes - Examples

- Hamming Codes - a class of single error correcting codes.
- Reed Solomon Codes.

Example
The (7, 4) Hamming Code

\[ G = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 \\
\end{pmatrix}. \]
Error Correcting Capability of a Block Code

- A block code can correct \( t \) errors if and only if 'Hamming' balls of size \( t \) around codewords don't intersect.
- Minimum distance of the code must be at least \( 2^t + 1 \).
- Linearity \( \implies \) minimum weight of the code is at least \( 2^t + 1 \).
Error Correcting Capability of a Block Code

A block code can correct $t$ errors if and only if 'Hamming' balls of size $t$ around codewords don't intersect.

Minimum distance of the code must be at least $2t + 1$.

Linearity $\Rightarrow$ Minimum weight of the code is at least $2t + 1$. 
A block code can correct $t$ errors if and only if ‘Hamming’ balls of size $t$ around codewords don’t intersect.

Minimum distance of the code must be at least $2t + 1$.

Linearity $\implies$ Minimum weight of the code is at least $2t + 1$. 
Decoding - The Parity Check Matrix

- The Parity Check Matrix: A full-rank $n - k \times n$ matrix such that $GH^T = 0$.

- For the Hamming Code:

$$H = \begin{pmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}.$$
Decoding

- Received vector \( y = c + e \).

- Compute \( s = y H^T = c H^T + e H^T = x G H^T + e H^T = e H^T \).

- Corresponding to any error vector of weight upto \( t \) there is an unique syndrome.
Syndrome Decoding

Syndrome decoding for errors of weight upto $t$.

1. Find the syndrome $s$

2. Find $e$ corresponding to $s$.

3. Find $c = r - e$. Map it back to $x$. 

## Types of Codes used in GNSS

### Table 2. Channel coding comparison

<table>
<thead>
<tr>
<th>Coding</th>
<th>NAV</th>
<th>CNAV</th>
<th>Galileo</th>
<th>CNAV-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Convolution</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CRC</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interleaving</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LDPC</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>BCH</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Convolutional Encoding

- Convolutional codes are used in applications that require good performance with low encoding complexity.

- Convolution codes have memory that utilises previous bits to encode or decode following bits (block codes are memoryless)
Convolutional Encoding

- $y_1[n] = x[n] \oplus x[n - 1] \oplus x[n - 2]$
- $y_2[n] = x[n] \oplus x[n - 2]$
- Rate $\frac{1}{2}$ convolutional encoder
- Constraint length for each input is 2
State Diagram

- State transitions are given by input/output
Example Encoding

Input: 010111001010001
Output: 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11
Brute Force Approach

- Going through the list of possible transmit sequences and comparing Hamming distance is highly complex
- A transmit sequence of $N$ bits has $2^N$ possible strings, exponential complexity
- Low Complexity Decoder: Viterbi Decoder - decoding on trellis
The branch metric for hard decision decoding. In this example, the receiver gets the parity bits 00.

Two of the branch metrics are 0, corresponding to the only states and transitions where the corresponding Hamming distance is 0.

Other non-zero branch metrics correspond to cases where there are bit errors.
Computing Path Metric

- Value of $PM[s, i]$ - total number of bit errors detected when comparing the received parity bits to the most likely transmitted message, considering all messages that could have been sent by the transmitter until time step $i$.

- If the transmitter is at state $s$ at time step $i + 1$, then it must have been in only one of two possible states at time step $i$, say $\alpha$ and $\beta$.

- Path Metric update is given by

$$PM[s, i+1] = \min(PM[\alpha, i] + BM[\alpha \rightarrow s], PM[\beta, i] + BM[\beta \rightarrow s])$$
Viterbi Decoding: Step 1
Viterbi Decoding: Step 2
Viterbi Decoding: Step 3

Showing only survivor paths
Viterbi Decoding: Step 4
To produce the message, start from final state with smallest path metric and word backwards and then reverse the bits.
Hard Decision Decoding

- Hard decision decoding digitizes the received voltage signals by comparing it to a threshold, before passing it to the decoder.

- Loss of Information

- 0.500001 and 0.99999 are both treated as “1” by the decoder even if it is more likely that 0.99999 is a “1”

- Hamming distance as branch metric
Soft Decision Decoding

- Soft Decision Decoding does not digitise the incoming samples prior to decoding

- If the convolutional code produces $p$ parity bits and $p$ corresponding analog samples are $v = v_1, v_2, \ldots, v_p$, a soft decision branch metric is given by

\[ BM_{\text{soft}}[u, v] = \sum_{i=1}^{p} (u_i - v_i)^2 \]
Thanks!