Chapter One

Introduction

1.1-Preface:

In telecommunication systems, 4G is a successor to the third generation (3G) standards. A 4G system provides mobile ultra-broadband Internet access, for example to laptops with USB wireless modems, to smartphones, and to other mobile devices. Conceivable applications include amended mobile web access, IP telephony, gaming services, high-definition mobile TV, video conferencing, 3D television, and cloud computing.

Two 4G candidate systems are commercially deployed: the Mobile Worldwide Interoperability for Microwave Access (WiMax) standard (first used in South Korea in 2006), and the first-release Long Term Evolution (LTE) standard (in Oslo, Norway and Stockholm, Sweden since 2009). The next figure shows the wireless technology migration [1].

![Figure 1.1: The Wireless Technology Migration](source)


[2]
The channel coding refers to the class of signal transformations designed to improve communications performance by enabling the transmitted signals to better withstand the effects of various channel impairments, such as noise, interference and fading. The channel coding is considered as an important signal processing operation which provides a reliable transmission of information over channel.

It is used mainly to minimize the effect of channel impairments by two basic operations, error detection and error correction [1].

1.2-Problem Definition:

The channel noise, interference, distortion and fading; all that can effect and produce errors in the information sent over mobile radio channel.

![Figure 1.2: Down Link Interference [3]](image1)

![Figure 1.3: Up Link Interference [3]](image2)
1.3-Objectives:
- Increase the signal to noise ratio ( \( \frac{S}{N} \))
- Decrease the bit error rate (BER)

1.4- Research Methodology:
- Study 4 G
- Understand the channel coding techniques
- Simulate the channel coding techniques and show the results
- Analysis the obtain results
- Compare between this techniques and dedicate the best to use

1.5-Thesis Outlines:
This research consists of the following chapters

**Chapter One:** includes an introduction, the problem definition, and the objectives.

**Chapter Two:** gives background about the channel coding beyond 4G.

**Chapter Three:** discusses the performance analysis of the channel coding.

**Chapter Four:** presents the simulation description of the channel coding techniques and discuss the results.

**Chapter Five:** contains the conclusion and recommendation.
Chapter Two

Literature Review

2.1 Introduction

The cellular wireless communications industry witnessed tremendous growth in the past decade with over four billion wireless subscribers worldwide. The first generation (1G) analog cellular systems supported voice communication with limited roaming. The second generation (2G) digital systems promised higher capacity and better voice quality than did their analog counterparts. Moreover, roaming became more prevalent thanks to fewer standards and common spectrum allocations across countries particularly in Europe. The two widely deployed second-generation (2G) cellular systems are GSM (global system for mobile communications) and CDMA (code division multiple access). As for the 1G analog systems, 2G systems were primarily designed to support voice communication. In later releases of these standards, capabilities were introduced to support data transmission. However, the data rates were generally lower than that supported by dial-up connections. The ITU-R initiative on IMT-2000 (international mobile telecommunications 2000) paved the way for evolution to 3G. A set of requirements such as a peak data rate of 2 Mb/s and supports for vehicular mobility were published under IMT-2000 initiative. Both the GSM and CDMA camps formed their own separate 3G partnership projects (3GPP and 3GPP2, respectively) to develop IMT-2000 compliant standards based on the CDMA technology. The 3G standard in 3GPP is referred to as Wideband CDMA (WCDMA) because it uses a larger 5MHz bandwidth relative to 1.25MHz bandwidth used in 3GPP2’s cdma2000 system. The
3GPP2 also developed a 5MHz version supporting three 1.25MHz subcarriers referred to as cdma2000-3x. In order to differentiate from the 5MHz cdma2000-3x standard, the 1.25MHz system is referred to as cdma2000-1x or simply 3G-1x.

The first release of the 3G standards did not fulfill its promise of high-speed data transmissions as the data rates supported in practice were much lower than that claimed in the standards. A serious effort was then made to enhance the 3G systems for efficient data support. The 3GPP2 first introduced the HRPD (high rate packet data) [2] system that used various advanced techniques optimized for data traffic such as channel sensitive scheduling, fast link adaptation and hybrid ARQ, etc. The HRPD system required a separate 1.25MHz carrier and supported no voice service. This was the reason that HRPD was initially referred to as cdma2000-1xEVDO (evolution data only) system. The 3GPP followed a similar path and introduced HSPA (high speed packet access) [3] enhancement to the WCDMA system. The HSPA standard reused many of the same data-optimized techniques as the HRPD system. A difference relative to HRPD, however, is that both voice and data can be carried on the same 5MHz carrier in HSPA. The voice and data traffic are code multiplexed in the downlink. In parallel to HRPD, 3GPP2 also developed a joint voice data standard that was referred to as cdma2000-1xEVDV (evolution data voice) [4]. Like HSPA, the cdma2000-1xEVDV system supported both voice and data on the same carrier but it was never commercialized. In the later release of HRPD, VoIP (Voice over Internet Protocol) capabilities were introduced to provide both voice and data service on the same carrier. The two 3G standards namely HSPA and HRPD were finally able to fulfill the 3G
promised and have been widely deployed in major cellular markets to provide wireless data access.

2.2- Beyond 3G systems:

While HSPA and HRPD systems were being developed and deployed, IEEE 802 LMSC (LAN/MAN Standard Committee) introduced the IEEE 802.16e standard [5] for mobile broadband wireless access. This standard was introduced as an enhancement to an earlier IEEE 802.16 standard for fixed broadband wireless access. The 802.16e standard employed a different access technology named OFDMA (orthogonal frequency division multiple access) and claimed better data rates and spectral efficiency than that provided by HSPA and HRPD. Although the IEEE 802.16 family of standards is officially called Wireless MAN in IEEE, it has been dubbed WiMAX (worldwide interoperability for microwave access) by an industry group named the WiMAX Forum. The mission of the WiMAX Forum is to promote and certify the compatibility and interoperability of broadband wireless access products. The WiMAX system supporting mobility as in IEEE 802.16e standard is referred to as Mobile WiMAX. In addition to the radio technology advantage, MobileWiMAX also employed a simpler network architecture based on IP protocols.

The introduction of Mobile WiMAX led both 3GPP and 3GPP2 to develop their own version of beyond 3G systems based on the OFDMA technology and network architecture similar to that in Mobile WiMAX. The beyond 3G system in 3GPP is called evolved universal terrestrial radio access (evolved UTRA) [6] and is also widely referred to as LTE (Long-Term Evolution) while 3GPP2’s version is called UMB (ultra mobile broadband) [6] as depicted in Figure 2.1. It should be noted that all three beyond 3G systems namely Mobile WiMAX,
2.3 -Evolution to 4G:

The radio-interface attributes for Mobile WiMAX and UMB are very similar to those of LTE given in Table 1.1. All three systems support flexible bandwidths, FDD/TDD duplexing, OFDMA in the downlink and MIMO schemes. There are a few differences such as uplink in LTE is based on SC-FDMA compared to OFDMA in Mobile WiMAX and UMB. The performance of the three systems is therefore expected to be similar with small differences.

Similar to the IMT-2000 initiative, ITU-R Working Party 5D has stated requirements for IMT-advanced systems. Among others, these requirements include average downlink data rates of 100 Mbit/s in the wide area network, and up to 1 Gbit/s for local access or low mobility scenarios. Also, at the World Radiocommunication Conference 2007 (WRC-2007), a maximum of a 428MHz new spectrum is identified for IMT systems that also include a 136MHz spectrum allocated on a global basis.
Both 3GPP and IEEE 802 LMSC are actively developing their own standards for submission to IMT-advanced. The goal for both LTE-advanced [8] and IEEE 802.16m [9] standards is to further enhance system spectral efficiency and data rates while supporting backward compatibility with their respective earlier releases. As part of the LTE-advanced and IEEE 802.16 standards developments, several enhancements including support for a larger than 20MHz bandwidth and higher-order MIMO are being discussed to meet the IMT-advanced requirements.

2.4- Channel coding:

In the engineering sense, coding can be classified into four areas:

- Encryption: to encrypt information for security purpose.
- Data compression: to reduce space for the data stream.
- Data translation: to change the form of representation of the information so that it can be transmitted over a communication channel.
- Error control: to encode a signal so that error occurred can be detected and possibly corrected.

The main aim of any communication schemes is to provide error-free data transmission. In a communication system, information can be transmitted by analog or digital signals. For analog means, the amplitude of the signal reflects the information of the source, whereas for digital case, the information will first be translated into a stream of ‘0’ and ‘1’. Then two different signals will be used to represent ‘0’ and ‘1’ respectively. As can be referred to the following illustration, the main advantage of using digital signal is that errors introduced by noise during the transmission can be detected and possibly corrected. For communication using cables, the
random motion of charges in conducting (e.g. resistors), known as thermal noise, is the major source of noise. For wireless communication channels, noise can be introduced in various ways. In the case of mobile phones, noise includes the signals sent by other mobile phone users in the system.

Figure 2.2: The flow of a simple digital communication system.

2.5 -Types of Channel Codes:

There are two main types of channel codes, namely block codes and convolutional codes. There are many differences between block codes and convolutional codes. Block codes are based rigorously on finite field arithmetic and abstract algebra. They can be used to either detect or correct errors. Block codes accept a block of k information bits and produce a block of n coded bits. By predetermined rules, n-k redundant bits are added to the k information bits to form the n coded bits. Commonly, these codes are referred to as (n, k) block codes. Some of the commonly used block codes are Hamming codes, Golay codes, BCH codes, and Reed Solomon codes (uses non binary symbols).
There are many ways to decode block codes and estimate the k information
bits. Convolutional codes are one of the most widely used channel codes in
practical communication systems. These codes are developed with a
separate strong mathematical structure and are primarily used for real time
error correction. Convolutional codes convert the entire data stream into one
single code word. The encoded bits depend not only on the current k input
bits but also on past input bits. The main decoding strategy for
convolutional codes is based on the widely used Viterbi algorithm
Error control coding is a method to detect and possibly correct errors
by introducing redundancy to the stream of bits to be sent to the channel. The
Channel Encoder will add bits to the message bits to be transmitted
systematically. After passing through the channel, the Channel decoder will
detect and correct the errors. A simple example is to send ‘000’ (‘111’
correspondingly) instead of sending only one ‘0’ (‘1’ correspondingly) to the
channel. Due to noise in the channel, the received bits may become ‘001’.
But since either ‘000’ or ‘111’ could have been sent. By majority logic
Decoding scheme, it will be decoded as ‘000’ and therefore the message has
been a ‘0’.
In general the channel encoder will divides the input message bits into
blocks of k message bits and replaces each k message bits block with a n-
bit code word by introducing (n-k) check bits to each message block. Some
major codes include the Block Codes and Convolutional Codes.
2.5.1 - Block Codes:

Denoted by \((n, k)\) a block code is a collection of code words each with length \(n\), \(k\) information bits and \(r = n - k\) check bits. It is linear if it is closed under addition mod 2. A Generator Matrix \(G\) (of order \(k \times n\)) is used to generate the code.

\[
G = [I_K \quad P]_{K \times n} (2.1)
\]

Where \(I_K\) is the \(k \times k\) identity matrix and \(P\) is a \(k \times (n - k)\) matrix selected to give desirable properties to the code produced. For example, denote \(D\) to be the message, \(G\) to be the generator matrix, \(C\) to be code word. For

\[
G = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}
\]

We get the collection of code words:

<table>
<thead>
<tr>
<th>Messages (D)</th>
<th>Code words (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 1 1 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 0 1 0 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 1 0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0 0 1 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 1 1 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0 1 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1 0 0 0</td>
</tr>
</tbody>
</table>

In particular when \(D = [0 1 1]\),

\[
C = DG = [0 1 1] \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix} = [0 1 1 0 1 1]
\]
Now define the Parity Check Matrix to be
\[ H = [p^T I_{n-K}]_{(n-K) \times n} \] (2.2)

As long as the code word \( C \) is generated by \( G \), the product \( CH^T = 0 \).

Denote the received code word after passing through the channel be \( R \). It is made up of the original code word \( C \) and error bits \( E \) from the channel. Further define the Error Syndrome to be
\[ S = RH^T \] (2.3)

Then,
\[ R = C + E \] (2.4)

\[ S = RH^T = (C + E)H^T = CH^T + EH^T = EH^T \] (2.5)

If \( S = 0 \), \( R \equiv C \) and \( D \) is the first \( k \) bits of \( R \). If \( S \neq 0 \) and \( S \) is the \( j^{th} \) row of \( H^T \), then it implies an error occurs in the \( j^{th} \) position of \( R \).

To illustrate, suppose after passing through the channel, the received code word is \( R = [0 \ 1 \ 1 \ 1 \ 1 \ 0] \).

Thus \( RHT = [0 \ 1 \ 1 \ 1 \ 1 \ 0]H^T = [1 \ 0 \ 1] \) which is the second row of \( H^T \). Thus it implies there exists an error in the second bit of the received code word. So we can correct it and detect that \([0 \ 0 \ 1 \ 1 \ 1 \ 0] \) should have been sent.

\[ \textbf{2.5.2 - Convolutional code:} \]

In telecommunication, a convolutional code is a type of error-correcting code in which

- Each m-bit information symbol (each m-bit string) to be encoded is transformed into an n-bit symbol, where \( m/n \) is the code rate \( (n \geq m) \) and
• The transformation is a function of the last k information symbols, where k is the constraint length of the code.

Convolutional codes are used extensively in numerous applications in order to achieve reliable data transfer, including digital video, radio, mobile communication, and satellite communication. These codes are often implemented in concatenation with a hard-decision code, particularly Reed Solomon. Prior to turbo codes, such constructions were the most efficient, coming closest to the Shannon limit.

2.5.2.1 Convolutional encoding:

To convolutionally encode data, start with k memory registers, each holding 1 input bit. Unless otherwise specified, all memory registers start with a value of 0. The encoder has n modulo-2 adders (a modulo 2 adder can be implemented with a single Boolean XOR gate, where the logic is: 0+0 = 0, 0+1 = 1, 1+0 = 1, 1+1 = 0), and n generator polynomials (one for each adder (see figure below). An input bit m1 is fed into the leftmost register. Using the generator polynomials and the existing values in the remaining registers, the encoder outputs n bits. Now bit shift all register values to the right (m1 moves to m0, m0 moves to m-1) and wait for the next input bit. If there are no remaining input bits, the encoder continues output until all registers have returned to the zero state.

The figure below is a rate 1/3 (m/n) encoder with constraint length (k) of 3. Generator polynomials are G1 = (1,1,1), G2 = (0,1,1), and G3 = (1,0,1). Therefore, output bits are calculated (modulo 2) as follows:

\[ n1 = m1 + m0 + m-1 \]
n2 = m0 + m-1

n3 = m1 + m-1.

Figure 2.3: Rate 1/3 non-recursive, non-systematic convolutional encoder with constraint length 3

Recursive and non-recursive codes

The encoder on the picture above is a non-recursive encoder. Here's an example of a recursive one:
One can see that the input being encoded is included in the output sequence too (look at the output 2). Such codes are referred to as systematic; otherwise the code is called non-systematic.

Recursive codes are almost always systematic and, conversely, non-recursive codes are non-systematic. It isn't a strict requirement, but a common practice.

**Impulse response, transfer function, and constraint length**

A convolutional encoder is called so because it performs a convolution of the input stream with the encoder's impulse responses:

\[
Y_i^j = \sum_{k=0}^{\infty} h_k^j x_{i-k} \tag{2.6}
\]

Where \(x\) is an input sequence, \(Y^j\) is a sequence from output \(j\) and \(h^j\) is an impulse response for output \(j\).
A convolutional encoder is a discrete linear time-invariant system. Every output of an encoder can be described by its own transfer function, which is closely related to the generator polynomial. An impulse response is connected with a transfer function through Z-transform.

Transfer functions for the first (non-recursive) encoder are:

\[
H_1(z) = 1 + z^{-1} + z^{-2},
\]
\[
H_2(z) = z^{-1} + z^{-2},
\]
\[
H_3(z) = 1 + z^{-2}.
\]

Transfer functions for the second (recursive) encoder are:

\[
H_1(z) = \frac{1 + z^{-1} + z^{-3}}{1 - z^{-2} - z^{-3}},
\]
\[
H_2(z) = 1.
\]

Define \( m \) by

\[
m \triangleq \max_{i} \text{polydeg}_i \left( H_i \left( \frac{1}{z} \right) \right) \tag{2.7}
\]

Where, for any rational function

\[
f(z) = \frac{P(z)}{Q(z)} \tag{2.8}
\]

\( \text{polydeg}(f) = \max(\text{deg}(P), \text{deg}(Q)) \) \tag{2}
Then \( m \) is the maximum of the polynomial degrees of the \( H_i \left( \frac{1}{Z} \right) \), and the constraint length is defined as \( K = m + 1 \). For instance, in the first example the constraint length is 3, and in the second the constraint length is 4.

**Trellis diagram**

A convolutional encoder is a finite state machine. An encoder with \( n \) binary cells will have \( 2^n \) states.

Imagine that the encoder (shown on Img.1, above) has '1' in the left memory cell (\( m_0 \)), and '0' in the right one (\( m_1 \)). (\( m_1 \) is not really a memory cell because it represents a current value). We will designate such a state as "10". According to an input bit the encoder at the next turn can convert either to the "01" state or the "11" state. One can see that not all transitions are possible for (e.g., a decoder can't convert from "10" state to "00" or even stay in "10" state).

All possible transitions can be shown as below:

![Trellis diagram](image)

*Figure 2.5: A trellis diagram for the encoder on Figure 2.3.*
An actual encoded sequence can be represented as a path on this graph. One valid path is shown in red as an example.

This diagram gives us an idea about decoding: if a received sequence doesn't fit this graph, then it was received with errors, and we must choose the nearest correct (fitting the graph) sequence. The real decoding algorithms exploit this idea.

**Free distance and error distribution**

The free distance \( d \) is the minimal Hamming distance between different encoded sequences. The correcting capability \( t \) of a convolutional code is the number of errors that can be corrected by the code. It can be calculated as

\[
t = \left\lfloor \frac{d-1}{2} \right\rfloor \quad (2.10)
\]

Since a convolutional code doesn't use blocks, processing instead a continuous bitstream, the value of \( t \) applies to a quantity of errors located relatively near to each other. That is, multiple groups of \( t \) errors can usually be fixed when they are relatively far apart.

Free distance can be interpreted as the minimal length of an erroneous "burst" at the output of a convolutional decoder. The fact that errors appear as "bursts" should be accounted for when designing a concatenated code with an inner convolutional code. The popular solution for this problem is to
interleave data before convolutional encoding, so that the outer block (usually Reed-Solomon) code can correct most of the errors.

2.5.2.2 Decoding convolutional codes

Several algorithms exist for decoding convolutional codes. For relatively small values of k, the Viterbi algorithm is universally used as it provides maximum likelihood performance and is highly parallelizable.

Longer constraint length codes are more practically decoded with any of several sequential decoding algorithms, of which the Fano algorithm is the best known. Unlike Viterbi decoding, sequential decoding is not maximum likelihood but its complexity increases only slightly with constraint length, allowing the use of strong, long-constraint-length codes. Such codes were used in the Pioneer program of the early 1970s to Jupiter and Saturn, but gave way to shorter, Viterbi-decoded codes, usually concatenated with large Reed-Solomon error correction codes that steepen the overall bit-error-rate curve and produce extremely low residual undetected error rates.

Both Viterbi and sequential decoding algorithms return hard decisions: the bits that form the most likely codeword. An approximate confidence measure can be added to each bit by use of the Soft output Viterbi algorithm. Maximum a posteriori (MAP) soft decisions for each bit can be obtained by use of the BCJR algorithm.
Popular convolutional codes

An especially popular Viterbi (nurul izyan) -decoded convolutional code, used at least since the Voyager program has a constraint length \( k \) of 7 and a rate \( r \) of 1/2.

- Longer constraint lengths produce more powerful codes, but the complexity of the Viterbi algorithm increases exponentially with constraint lengths, limiting these more powerful codes to deep space missions where the extra performance is easily worth the increased decoder complexity.
- Mars Pathfinder, Mars Exploration Rover and the Cassini probe to Saturn use a \( k \) of 15 and a rate of 1/6; this code performs about 2 dB better than the simpler \( k=7 \) code at a cost of 256× in decoding complexity (compared to Voyager mission codes).

Punctured convolutional codes

Puncturing is a technique used to make a \( m/n \) rate code from a "basic" low-rate (e.g., 1/n) code. It is reached by deletion of some bits in the encoder output. Bits are deleted according to a puncturing matrix. The following puncturing matrices are the most frequently used:

Table 2.1: The Puncturing Matrix

<table>
<thead>
<tr>
<th>Code rate</th>
<th>Puncturing matrix</th>
<th>Free distance (for NASA standard K=7 convolutional code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

20
For example, if we want to make a code with rate 2/3 using the appropriate matrix from the above table, we should take a basic encoder output and transmit every second bit from the first branch and every bit from the second one. The specific order of transmission is defined by the respective communication standard.

Punctured convolutional codes are widely used in the satellite communications, for example, in INTELSAT systems and Digital Video Broadcasting.

Punctured convolutional codes are also called "perforated".

2.5.3 - Turbo codes:

Simple Viterbi-decoded convolutional codes are now giving way to turbo codes, a new class of iterated short convolutional codes that closely approach the theoretical limits imposed by Shannon's theorem with much less decoding complexity than the Viterbi algorithm on the long
concatenation with an outer algebraic code (e.g., Reed-Solomon) addresses the issue of error floors inherent to turbo code designs.

2.5.3.1 - Turbo encoder:
The fundamental of Turbo encoder is built using two identical recursive systematic convolutional (RSC) code with parallel concatenation as shown in the figure 2.6[2]

![Turbo Code Encoder](image)

**Figure 2.6: Turbo Code Encoder**

Where the encoders are based on RSC (Recursive Systematic Convolutional) codes and their generator polynomial is given by $G=[1, g_0/g_1]$, where $g_0=[1011]$ (feedback) and $g_1=[1101]$ (feed forward).

The structure of the Turbo encoder used can be seen in Fig. 2.7.
As can be seen in Fig 7, the output Turbo encoder consists of three parts, a systematic bit and two parity bits. The systematic bit \((X_k)\) is the untouched input bit. The first parity bit \((Z_k)\) is the output of the first convolutional encoder with the original input \((C_k)\) as input and the second parity bit \((Z_k')\) is the output of the second convolutional encoder after interleaving (by the Turbo code internal interleaver) of the input bit \((C_k')\) as its input. For trellis termination the tail-bits \((X_k')\) are inserted.

**Turbo encoders internal contention free interleaver:**

The internal contention-free interleaver is one of the key parts of the Turbo encoder. The main difference between the Turbo encoders in LTE and UMTS is that the interleaver is in contrast to the one in UMTS contention-free. Contentions occur, when parallel working processes try to write or read to/from the same memory address simultaneously. Because the two SISO (Soft-Input Soft-Output) MAP decoder engines of the Turbo
decoder use such processes, the contention-free concept becomes survival for designing the Turbo encoders internal interleaver efficiently.

A contention-free interleaver \( \pi(i), 0 \leq i \leq K \) should satisfy the following inequality, where \( W \) is the window size and \( 0 \leq \nu W, u_1 \geq 0, u_2 < M \) for all \( u_1 = u_2 \):

\[
\left\lfloor \frac{\psi(u1W+\nu)}{W} \right\rfloor \neq \left\lfloor \frac{\psi(u2W+\nu)}{W} \right\rfloor \tag{2.11}
\]

The above inequality satisfies for both interleaver \( \psi = \pi \) and \( (de) \)interleaver \( \psi = \pi^{-1} \) and indicates, that the memory addresses (both sides of the inequality), accessed by \( M \) processes on the \( v^{th} \) step must be different.

There were two candidates for internal interleaver: Almost Regular Permutation (ARP) and Quadrature Permutation Polynomial (QPP), which are very similar. However QPP is chosen for LTE because it offers more parallelism. The QPP interleaver for a block size of \( K \) is defined as following:

\[
\pi(i) = (f_1 i + f_2 i^2) \mod K \tag{2.12}
\]

Where \( i \) is the input index and \( \pi(i) \) the output index and \( f_1 \) and \( f_2 \) are permutation parameters, which can be get from the standard.

2.5.3.2 - Turbo decoder:

The Turbo decoder is based on two SISO (Soft-Input Soft-Output) decoders, which work together in an iterative manner. Each SISO decode has two inputs, namely a normal input and an a-priori Log Likelihood Ratio (LLR) input, and two outputs, an a-posteriori LLR and an extrinsic LLR. In each iteration the (de)interleaved version of the extrinsic
output of a decoder is used as the a-priori information for the other decoder. Typically after 4 to 8 iterations, the a-posteriori LLR output can be used to obtain the final hard decision estimates of the information bits. The structure of the Turbo decoder can be seen in Fig 2.8.

Figure 2.8: Turbo decoder
Chapter 3

Channel Coding in 4G

3.1 - Turbo Code:

The Turbo code (Berrou et al., 1993; Berrou et al., 1996) has become one of the most important research topics in coding theory since its discovery in 1993. The astounding performance of Turbo code has attracted a great deal of interest in the research activity in the area of iterative error correction codes. Due to its excellent error correction performance, many communication standards have chosen Turbo codes as the Forward Error Correction (FEC) codes, such as CDMA-2000, W-CDMA, DVB-RCS, HSDPA, UMTS, IEEE 802.16e WiMax, and 3GPP LTE. Turbo codes can be categorized into two classes: binary Turbo codes and non-binary Turbo codes. For example, Turbo codes in CDMA, HSDPA, UMTS and 3GPP LTE are binary types of Turbo codes, whereas Turbo codes in IEEE 802.16e and DVB-RCS are double-binary types of Turbo codes.

Table 3.1 summarizes some of the Turbo codes in practice (Berrou, 2003). As we can see, there are many similarities between the Turbo codes employed in different standards. This motivates the design of a unified and flexible Turbo decoder which can support multiple standards. Without loss of generality, we will mainly focus on the Turbo codes defined in 3GPP LTE and WiMax in the following analysis. Note that these analyses can be applied to other systems directly because the encoder polynomials are same.
### Table 3.1: Some applications of Turbo codes

<table>
<thead>
<tr>
<th>Application</th>
<th>Code structure</th>
<th>Polynomials</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDMA, WCDMA, UMTS, LTE</td>
<td>8-state binary</td>
<td>13, 15, 17</td>
</tr>
<tr>
<td>WiMax, DVB-RCS</td>
<td>8-state double-binary</td>
<td>15, 13</td>
</tr>
</tbody>
</table>

### 3.2- Binary Turbo Code in 3GPP LTE Standard:

Turbo coding scheme in 3GPP LTE standard (3GPP TS 36.212, 2008) is a parallel concatenated convolutional code (PCCC) with two 8-state constituent encoders and one quadratic permutation polynomial (QPP) interleaver. The coding rate of the Turbo code is 1/3. The structure of the Turbo encoder is shown in Figure 3.1.

![Figure 3.1: Structure of rate 1/3 Turbo encoder in 3GPP LTE](image)
As seen in the figure, a Turbo encoder consists of two binary convolutional encoders connected by an interleaver. The basic coding rate is 1/3 which means N data bits will be coded into 3N data bits. The transfer function of the 8-state constituent code for PCCC is:

\[ G(D) = \begin{bmatrix} 1, \frac{g_1(D)}{g_0(D)} \end{bmatrix} \]  

(3.1)

Where

\[ g_0(D) = 1 + D^2 + D^3 \]  

(3.2)

\[ g_1(D) = 1 + D + D^3 \]  

(3.3)

The initial value of the shift registers of the 8-state constituent encoders shall be all zeros when starting to encode the input bits. Trellis termination is performed by taking the tail bits from the shift register feedback after all information bits are encoded. Tail bits are padded after the encoding of information bits.

The function of the Interleaver is to take each incoming block of N data bits and shuffle them in a pseudo-random manner. One of the new features in the 3GPP LTE Turbo encoder is its quadratic permutation polynomial (QPP) internal interleaver. We will see later that this QPP interleaver is the key component enabling parallel decoding of turbo codes.

### 3.3 - Double Binary Turbo Code in IEEE 802.16e WiMax Standard:

The convolutional Turbo encoder for the IEEE 802.16e standard (IEEE Std 802.16, 2004) is depicted in Figure 3.2. It uses a double binary circular recursive systematic convolutional code. Data couples (A, B) rather than a single bit sequence, are fed to the circular recursive systematic convolutional
encoder twice, and four parity bits (Y1, W1) and (Y2, W2) are generated in the natural order and the interleaved order, respectively. The encoder polynomials are described in binary symbol notation as follows:

For the feedback branch: \(1 + D + D^3 (3.4)\)

For the Y parity bit: \(1 + D^2 + D^3 (3.5)\)

For the W parity bit: \(1 + D^3 (3.6)\)

The tail-biting Trellis termination scheme is used as opposed to inserting extra tail bits. In this termination scheme, the start state of the trellis equals to the end state of the trellis. Therefore, a pre encoding operation has to be performed to determine the start state. This is not a complex problem because the encoding process can be performed at a much higher rate. A symbol-wise almost regular permutation (ARP) interleaver is used in the WiMax standard, which can enable parallel decoding of double binary Turbo codes.

![Diagram](image)

**Figure 3.2: Structure of rate 1/3 double binary Turbo encoder in IEEE 802.16e**
The decoding algorithm employed in the Turbo decoders is the maximum a posteriori (MAP) algorithm proposed by Bahel-Cocke-Jelniek-Raviv. So it is also called the BCJR algorithm.

The Turbo decoding concept is functionally illustrated in Figure 3.2. As discussed before, the decoding is based on the MAP algorithm and is usually calculated in the log domain (Robertson et al., 1995) to avoid multiplications and divisions. During the decoding process, each soft-in soft-output (SISO) decoder receives the intrinsic log-likelihood ratios (LLRs) from the channel and the extrinsic LLRs from the other constituent SISO decoder through interleaving (\(\Pi\)) or deinterleaving (\(\Pi^{-1}\)). The main task of the Turbo internal interleaver is to generate a permutation of the input data sequence that is as uncorrelated as possible. The randomness of the interleaver not only affects the decoding performance, but also leads to decoding latency because one SISO decoder must wait for the other SISO decoder to finish decoding before it can start the next iteration.

![Diagram of Turbo encoder and decoder](image)

**Figure 3.3: Basic structure of Turbo encoder and decoder**

An efficient representation of the Turbo decoding process is the trellis diagram which describes all the possible state transitions through a graph representation. Figure 3.4 shows a section of the trellis diagram for an 8-
state binary Turbo code, where the dashed edges correspond to input bit $u_k=0$, and solid edges correspond to $u_k=1$.

![Trellis diagram for an 8-state binary Turbo code](image)

**Figure 3.4: Trellis diagram for an 8-state binary Turbo code**

The MAP algorithm is an optimal symbol decoding algorithm that minimizes the probability of a symbol error. It computes the a posteriori probabilities (APPs) of the information bits given the received sequence. The MAP algorithm can be summarized as follows:

$$L(u_k^l) = \log \frac{p(u_k = +1|Y)}{p(u_k = -1|Y)} = \log \frac{\sum_{u_{k-1}^l} p(S_{k-1} = S^l, S_k = S, Y)}{\sum_{u_{k-1}^l} p(S_{k-1} = S^l, S_k = S, Y)}$$ (3.7)

For computing $p(S_{k-1} = S^l, S_k = S, Y)$, BCJR algorithm can be applied:

$$p(S_{k-1} = S^l, S_k = S, Y) = \alpha_{k-1}(s^l) \cdot \gamma_k(s^l, s) \cdot \beta_k(s)$$ (3.8)
Where $\alpha_K$ and $\beta_k$ are referred to forward and backward metrics and are computed as:

\[
\alpha_K(s) = \sum_{s'} \gamma_k(s', s) \cdot \alpha_{K-1}(s') \tag{3.9}
\]

\[
\beta_k(s') = \sum_s \gamma_k(s, s') \cdot \beta_{k+1}(s) \tag{3.10}
\]

In the above equations, $\gamma$ is the state transition probability and is computed as\[11\]:

\[
\gamma_k(s', s) = P(s|s') \cdot P(y_k|s', s) = P(u_k) \cdot P(y_k|u_k)
\]

\[
= C_k \exp \{ \frac{1}{2} u_k (L(u_k) + L_c y_k^s) + \frac{1}{2} L_c y_k^p x_k^p \} \tag{3.11}
\]

Where $C_k$ is a constant and will not affect the calculation of $L(u_k)$. $L_c = \frac{4E_s}{N_0} \cdot L(u_k)$ (3.12)

is the log-likelihood ratio of $u_k$ defined as:

\[
L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)} \tag{3.13}
\]

Now the a posteriori probability (APP) log-likelihood ratio (LLR) of the information bits can be expressed as:

\[
L(u_k^\text{APP}) = \log \frac{P(u_k = +1|Y)}{P(u_k = -1|Y)} = \log \frac{\sum_{u_k = +1} \alpha_{k-1} = (s') \gamma_k(s', s) \beta_k(s)}{\sum_{u_k = -1} \alpha_{k-1} = (s') \gamma_k(s', s) \beta_k(s)} \tag{3.14}
\]
Chapter Four

Simulation and Results

In this section, the results are obtained from the mathematical expressions presented in the previous section in chapter three based on Matlab simulation. The Matlab code can be found in the appendix. These results are obtained of BPSK modulation for different block lengths.

4.1 - Environment and Parameters Assumptions:

Table 4.1 gives a list of main simulation parameters used through the simulation performances.

Table 4.1: simulation parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>BCJR</td>
</tr>
<tr>
<td>Code rate</td>
<td>$1/3$</td>
</tr>
<tr>
<td>Modulation</td>
<td>BPSK</td>
</tr>
<tr>
<td>Channel</td>
<td>AWGN</td>
</tr>
<tr>
<td>Iteration</td>
<td>8</td>
</tr>
</tbody>
</table>
4.2 -Simulation Results and Discussion:
The following results show the effect of the block length, at figure 4.1 doesn’t show the values of the signal to noise ratio and bit error rate clear because the block length is very small. At the other figures notice that; the first iteration near to theoretical values and 8th iteration show less BER and more $S/N$. 

Figure 4.1: Turbo decoder performance over AWGN channel for length =1E1
Figure 4.2: Turbo decoder performance over AWGN channel for length = 1E2
Figure 4.3: Turbo decoder performance over AWGN channel for length =2E3
Figure 4.4: Turbo decoder performance over AWGN channel for length =3E3
Figure 4.5: Turbo decoder performance over AWGN channel for length =1E4
Figure 4.6: Turbo decoder performance over AWGN channel for length =2E4
Figure 4.7: Turbo decoder performance over AWGN channel for length $=3E4$
Figure 4.8: Turbo decoder performance over AWGN channel for length = 4E4
Chapter Five

Conclusion and Recommendations

5.1- Conclusion:
The turbo coding performance improves with an increase in the block lengths. For larger block lengths the BER is found to be less for a given value of S/N. This result confirms the increasing of the iteration number decreasing the BER.

5.2 - Recommendations:
The researcher suggests the following recommendations for the operation process:

- In this research not used different types of modulation such as: QPSK and QAM.
- The types of decoding algorithms can be used to improve the performance of turbo code such as: Log MAP and Max Log MAP.
References

[1] http://4g-communication

techiques.blogspot.com/2013_03_01_archive.html


Rice University, USA Yuming Zhu Texas Instruments, USA Manish Goel

Texas Instruments – Turbo Decoder - First Edition- 2010

[13] Patel SnehaBhanubhai, Mary Grace Shajan, Upena D. Dalal


[16] Prof. Dr. Mehmet B. Baray - DOUBLE BINARY TURBO CODES ANALYSIS AND DECODER IMPLEMENTATION - Bilkent University - 2008


Appendix

%%% Turbo Code
% Encoder: RSC (Recursive Systematic Convolution)
% Decoder: BCJR iterative decoder

%%% Parameter declaration
close all; clear all; clc;
N=1E4; % Block length
X=floor(2*rand(1,N)); % Information bit generation
Interleaver=randperm(N); % Interleaver(random permutation of first N integers)
SNRdB=0:0.5:9; % SNR in dB
SNR=10.^(SNRdB/10); % SNR in linear scale
Iteration=8;
ber=zeros(length(SNR),Iteration); % Simulated BER(Each column corresponds to one iteration)

%%% Encoding
X_pi(1:N)=X(Interleaver(1:N)); % Interleaving input bits for RSC-1 encoder

C0=zeros(1,N); % Code Bit for encoder RSC-0
C1=zeros(1,N); % Code Bit for encoder RSC-1
for i=1:N
  k = i;
  while (k >= 1)
    C0(i) = xor ( C0(i),X(k) );
    C1(i) = xor ( C1(i),X_pi(k) );
    k=k-2;
  end
end
P0 = xor (X,[0,C0(1:end-1)]);
P1 = xor (X_pi,[0,C1(1:end-1)]);
Input_matrix=2*[0,1;0,1;0,1;0,1]-1;       %First column represents
input=0 and second column represents input=1
Each row represents state 00,10,01 and 11 respectively
Parity_bit_matrix=2*[0,1;1,0;0,1;1,0]-1;   %Parity bits corresponding to
inputs of above matrix

mod_code_bit0=2*X-1;       %Modulating Code Bits using BPSK
Modulation
mod_code_bit1=2*P0-1;
mod_code_bit2=2*P1-1;

dlg = ProgressDialog();
dlg.FractionComplete = 0;
dlg.StatusMessage = sprintf('Encoding completed...');

%% Decoding
for k = 1:length(SNR)       %Simulation starts here
    R0=sqrt(SNR(k))*mod_code_bit0+randn(1,N);   % Received Codebits
    Corresponding to input bits
    R1=sqrt(SNR(k))*mod_code_bit1+randn(1,N);   % Received Codebits
    Corresponding to parity bits of RSC-0
    R2=sqrt(SNR(k))*mod_code_bit2+randn(1,N);   % Received Codebits
    Corresponding to parity bits of RSC-1

    R0_pi(1:N)=R0(Interleaver(1:N));   %Interleaving received
codebits corresponding to input bits to be used by RSC-1

    BCJR=0;       %First iteration will be done by BCJR-0
    Apriori=ones(2,N);       %First row for prob. of i/p 0 and second row for
    prob. of i/p 1
    Apriori=Apriori*0.5;       %Initializing all apriori to 1/2

    for iter=1:Iteration       %Iterative process starts here
        if BCJR==0   %If BCJR is 0 then pass R0 and R1 to calculate GAMMA
            GAMMA=gamma_1(Apriori,N,Input_matrix,Parity_bit_matrix,R0,R1,SNR(k));
        else%If BCJR is 1 then pass R0_pi and R2 to calculate GAMMA
            GAMMA=gamma_1(Apriori,N,Input_matrix,Parity_bit_matrix,R0,R1,SNR(k));
        end
    end
end

46
GAMMA=gamma_1(Apriori,N,Input_matrix,Parity_bit_matrix,R0_pi,R2,SNR(k));
end

ALPHA=alpha_1(GAMMA,N); %Calculation of ALPHA at each stage using GAMMA and ALPHA of previous stage
BETA=beta_1(GAMMA,N); %Calculation of BETA at each stage using GAMMA and BETA of next stage

%Calculating LAPPR using ALPHA,BETA and GAMMA
 [~,~,LAPPR_1]=lappr(ALPHA,BETA,GAMMA,N);

decoded_bits=zeros(1,N);
decoded_bits(LAPPR_1>0)=1; %Decoding is done using LAPPR values

if BCJR==0 %If the decoder is BCJR-0 then
 ber(k,iter)=sum(abs((decoded_bits-X))); %calculate BER using input X
 lappr_2(1:N)=LAPPR_1(Interleaver(1:N)); %Interleave the LAPPR values and pass to BCJR-1
else %If the decoder is BCJR-1 then
 ber(k,iter)=sum(abs((decoded_bits-X_pi))); %calculate BER using input X_pi
 lappr_2(Interleaver(1:N))=LAPPR_1(1:N); %Re-interleave the LAPPR values and pass to BCJR-0
end
LAPPR_1=lappr_2;
ber(ber==1)=0; %Ignoring 1 bit error
Apriori(1,1:N)=1./(1+exp(LAPPR_1)); %Apriori corresponding to input 0
Apriori(2,1:N)=exp(LAPPR_1)./(1+exp(LAPPR_1)); %Apriori corresponding to input 1
BCJR=~BCJR; %Changing the state of the decoder for the next iteration

end %One iteration ends here
u = round(k/length(SNR) * 100);
dlg.FractionComplete = k/length(SNR);
dlg.StatusMessage = sprintf('%d%% Decoding completed',u);
end
ber=ber/N;
figure;
%%% Plots for simulated BER
dlg.StatusMessage = sprintf('Done!');
semilogy(SNRdB,ber(:,1),'k--',,'linewidth',2.0);
hold on
semilogy(SNRdB,ber(:,2),'m-o',,,'linewidth',2.0);
hold on
semilogy(SNRdB,ber(:,3),'b-<',,,'linewidth',2.0);
hold on
semilogy(SNRdB,ber(:,4),'r-<',,,'linewidth',2.0);
hold on
semilogy(SNRdB,ber(:,5),'c--',,,'linewidth',2.0);
hold on
semilogy(SNRdB,ber(:,6),'r-.',,,'linewidth',2.0);
hold on
semilogy(SNRdB,ber(:,7),'g--',,,'linewidth',2.0);
hold on
semilogy(SNRdB,ber(:,8),'y--',,,'linewidth',2.0);
hold on
%%% Theoretical expression for BER for corresponding convolution code
BER=zeros(1,length(SNR));
for j=1:10
    BER=BER+(2^j)*(j)*qfunc(sqrt((j+4)*SNR));
end
semilogy(SNRdB,BER,'c-',,,'linewidth',2.0)
title('Turbo decoder performance over AWGN channel for BPSK modulated symbols');
xlabel('SNR(dB)');ylabel('BER');
legend('1st Iteration','2nd Iteration','3rd Iteration','4th Iteration','Theoretical Bound');
grid on
axis tight

%This function calculates BETA probabilities at each stage for all states, %using GAMMA probabilities obtained previously. Uses recursion formula for %BETA to calculate it for the previous stage. Each column is for states 00,10,01
%and 11 respectively. As we move backward in the block BETA will
become very
%small, as the 1st term corresponding to gamma can be very less(of the
order
%of 10\(^{-15}\)). Hence BETA will keep on decreasing and will become very
small.
%After some stages it will become exactly 0. So to avoid that we can
%multiply each BETA by 10\(^{-20}\) at a stage where they all become less
than
%10\(^{-20}\). As we need BETA in calculation of LAPPR. So scaling wont
affect the ratio

function [BETA]=beta_1(GAMMA,N)

    BETA=zeros(4,N);
%Initialization assuming the final stage to be 00
    BETA(1,N)=1;BETA(2,N)=0;BETA(3,N)=0;BETA(4,N)=0;

    j=2*N-1;
    for i=N-1:-1:1

        BETA(1,i)=(GAMMA(1,j)*BETA(1,i+1))+(GAMMA(1,j+1)*BETA(2,i+1))
        ;

        BETA(2,i)=(GAMMA(2,j)*BETA(3,i+1))+(GAMMA(2,j+1)*BETA(4,i+1))
        ;

        BETA(3,i)=(GAMMA(3,j)*BETA(2,i+1))+(GAMMA(3,j+1)*BETA(1,i+1))
        ;

        BETA(4,i)=(GAMMA(4,j)*BETA(4,i+1))+(GAMMA(4,j+1)*BETA(3,i+1))
        ;
        j=j-2;

        if (BETA(1,i)<10\(^{-20}\) && BETA(2,i)<10\(^{-20}\) &&
        BETA(3,i)<10\(^{-20}\) && BETA(4,i)<10\(^{-20}\) )
        BETA(:,i)=10\(^{20}\)*BETA(:,i);
        end
    end
end

This function calculates ALPHA probabilities at each stage for all states, using GAMMA probabilities obtained previously. Uses recursion formula for ALPHA to calculate it for the next stage. Each column is for states 00, 10, 01 and 11 respectively. As we move forward in the block ALPHA will become very less, as the 1st term corresponding to gamma can be very less (of the order of $10^{-15}$). Hence ALPHA will keep on decreasing and will become very small. After some stages it will become exactly 0. So to avoid that we can multiply each ALPHA by $10^{-20}$ at a stage where they all become less than $10^{-20}$. As we need ALPHA in calculation of LAPPR. So scaling won't affect the ratio

function [ALPHA]=alpha_1(GAMMA,N)

ALPHA=zeros(4,N);

%Initialization of alpha assuming first state to be 00
ALPHA(1,1)=1;ALPHA(2,1)=0;ALPHA(3,1)=0;ALPHA(4,1)=0;

j=1;
for i=2:N
    ALPHA(1,i)=((GAMMA(1,j)*ALPHA(1,i-1))+(GAMMA(3,j+1)*ALPHA(3,i-1)));
    ALPHA(2,i)=((GAMMA(3,j)*ALPHA(3,i-1))+(GAMMA(1,j+1)*ALPHA(1,i-1)));
    ALPHA(3,i)=((GAMMA(2,j)*ALPHA(2,i-1))+(GAMMA(4,j+1)*ALPHA(4,i-1)));
    ALPHA(4,i)=((GAMMA(4,j)*ALPHA(4,i-1))+(GAMMA(2,j+1)*ALPHA(2,i-1)));
    j=j+2;

if (ALPHA(1,i)<10^(-20) && ALPHA(2,i)<10^(-20) && ...
    ALPHA(3,i)<10^(-20) && ALPHA(4,i)<10^(-20) )
ALPHA(:,i)=10^20*ALPHA(:,i);  %Scaling Alpha if became very less
% Calculates LAPPR at each stage using ALPHA, BETA and GAMMA probabilities. Returns P(X(k)=0), P(X(k)=1) and LAPPR(k) for all k's

function [p_x0, p_x1, lappr_1] = lappr(ALPHA, BETA, GAMMA, N)
    p_x0 = zeros(1, N);
p_x1 = zeros(1, N);
lappr_1 = zeros(1, N);

    for i = 1:N
        p_x1(i) = (ALPHA(1,i)*GAMMA(1,2*i)*BETA(2,i)) + (ALPHA(2,i)*GAMMA(2,2*i)*BETA(4,i)) + ... 
                   (ALPHA(3,i)*GAMMA(3,2*i)*BETA(1,i)) + (ALPHA(4,i)*GAMMA(4,2*i)*BETA(3,i));

        p_x0(i) = (ALPHA(1,i)*GAMMA(1,2*i-1)*BETA(1,i)) + (ALPHA(2,i)*GAMMA(2,2*i-1)*BETA(3,i)) + ... 
                   (ALPHA(3,i)*GAMMA(3,2*i-1)*BETA(2,i)) + (ALPHA(4,i)*GAMMA(4,2*i-1)*BETA(4,i));

        lappr_1(i) = log(p_x1(i)/p_x0(i));
    end
end

% Root class for composite user controls.

classdef UIControl < handle
    properties (Abstract, Dependent)
        % The graphics handle that represents the entire user control.
        % This handle is used in setting the parent, position, measurement
        % unit, etc. properties of the control. In order to use MatLab's Handle

% Graphics semantics in a derived user interface control, you should
% query the Control property.
  Control;
end

properties (Dependent)
  NormalizedPosition;
  % The parent control of this UI control.
  Parent;
  % Units in which the Position property is expressed.
  % Note: this property is deliberately put before the Position property
  % to ensure proper initialization when both Units and Position are set
  % in a constructor argument list. For the constructor processes
  % arguments in the order in which they occur within the class
  % definition rather than the order in which they are specified, we
  % always get the desired behavior that Position is interpreted in terms
  % of the specified Units.
  Units = 'normalized';
  Position;
end

methods (Abstract)
% Forces instantiation of underlying MatLab Handle Graphics objects.
% Some dependent properties refer to MatLab Handle Graphics object
% properties in an early phase of object construction. These properties
% might fail if existence of objects they depend on cannot be
% guaranteed.
obj = Instantiate(obj, parent);
end

methods
  function obj = UIControl(arg, varargin)
    if nargin > 0 && iscalar(arg) && ishandle(arg)
      obj = obj.Instantiate(arg);
    else
      obj = obj.Instantiate();
    end
    if nargin > 0
      if iscalar(arg) && ishandle(arg)
        obj.Parent = arg;
        obj = constructor(obj, varargin{:});
      else
        if strcmp('figure', get(obj.Control, 'Type'))
          % Further code...
        end
      end
    end
end
obj.Parent = gcf;
end
obj = constructor(obj, arg, varargin{::});
end
end
end
end
end

function parent = get.Parent(obj)
parent = get(obj.Control, 'Parent');
end

function obj = set.Parent(obj, parent)
if ~strcmp(get(obj.Control, 'Type'), 'figure')  % figures do not require a parent
set(obj.Control, 'Parent', parent);
end
end

function units = get.Units(obj)
units = get(obj.Control, 'Units');
end

function obj = set.Units(obj, units)
set(obj.Control, 'Units', units);
end

function position = get.Position(obj)
position = get(obj.Control, 'Position');
end

function obj = set.Position(obj, position)
set(obj.Control, 'Position', position);
end

function obj = set.NormalizedPosition(obj, position)
units = get(obj.Control, 'Units');
if isempty(units)
units = 'normalized';
end
switch units
case 'normalized'
set(obj.Control, 'Position', position);
otherwise
set(obj.Control, 'Units', 'normalized');
set(obj.Control, 'Position', position);
set(obj.Control, 'Units', units);
end
end

function position = get.NormalizedPosition(obj)
units = get(obj.Control, 'Units');
if isempty(units)
units = 'normalized';
end
switch units
case 'normalized'
position = get(obj.Control, 'Position');
otherwise
set(obj.Control, 'Units', 'normalized');
position = get(obj.Control, 'Position');
set(obj.Control, 'Units', units);
end
end

function position = getpixelposition(obj)
position = getpixelposition(obj.Control);
end

function obj = setpixelposition(obj, position)
setpixelposition(obj.Control, position);
end
end
end

obj = constructor(obj, arg, varargin{:});
end
end
end

function parent = get.Parent(obj)
parent = get(obj.Control, 'Parent');
function obj = set.Parent(obj, parent)
if ~strcmp(get(obj.Control, 'Type'), 'figure')  % figures do not require a parent
    set(obj.Control, 'Parent', parent);
end
end

function units = get.Units(obj)
units = get(obj.Control, 'Units');
end

function obj = set.Units(obj, units)
set(obj.Control, 'Units', units);
end

function position = get.Position(obj)
position = get(obj.Control, 'Position');
end

function obj = set.Position(obj, position)
set(obj.Control, 'Position', position);
end

function obj = set.NormalizedPosition(obj, position)
units = get(obj.Control, 'Units');
if isempty(units)
    units = 'normalized';
end
switch units
    case 'normalized'
        set(obj.Control, 'Position', position);
    otherwise
        set(obj.Control, 'Units', 'normalized');
        set(obj.Control, 'Position', position);
        set(obj.Control, 'Units', units);
end
end
function position = get.NormalizedPosition(obj)
units = get(obj.Control, 'Units');
if isempty(units)
    units = 'normalized';
end
switch units
    case 'normalized'
        position = get(obj.Control, 'Position');
    otherwise
        set(obj.Control, 'Units', 'normalized');
        position = get(obj.Control, 'Position');
        set(obj.Control, 'Units', units);
end
end

function position = getpixelposition(obj)
position = getpixelposition(obj.Control);
end

function obj = setpixelposition(obj, position)
setpixelposition(obj.Control, position);
end

% Extracts size information from handle graphics position vector.
% Input arguments:
% dims:
%   a four-element position vector storing object left and bottom
%   coordinates, as well as object width and height
% Example:
%   [w,h] = position2size(get(gcf, 'Position'));
% See also: get, set

function [width,height] = position2size(dims)
width = dims(3);
height = dims(4);
% Displays a progress dialog to notify the user of an ongoing operation.
%
% Input arguments:
% h:
% graphics handle to a progress bar dialog, or empty if a new window is
% about to be created
% r:
% completion ratio, a number between 0.0 and 1.0, or empty if the
% completion state is indeterminate
% msg:
% a message string to be displayed in the window, or empty to display
% no message

function h = progressbar(h, r, msg, usejavactrl)

if nargin>= 1 && ~isempty(h)
    validateattributes(h, {'numeric'}, {'scalar'});
    assert(h ~= 0 && ishandle(h), ...
    'gui:waitdialog:ArgumentTypeMismatch', ...
    'First argument is expected to be a graphics handle.');
else
    h = [];
end
if nargin>= 2 && ~isempty(r)
    validateattributes(r, {'numeric'}, {'nonnegative','real','scalar'});
else
    r = [];
end
if nargin>= 3 && ~isempty(msg)
    validateattributes(msg, {'char'}, {'nonempty','row'});
    msgupdate = true;
elseif nargin>= 3
    msgupdate = true;
else
    msgupdate = false;
    msg = "";
end
if nargin>= 4
    assert(isempty(h), ...
    'gui:waitdialog:ArgumentTypeMismatch', ...
'Cannot change implementation once the dialog is constructed.';
if isempty(usejavactrl)
usejavactrl = true;
else
validateattributes(usejavactrl, {'logical'}, {'scalar'});
end
else
usejavactrl = true;
end

defaultmsg = 'Please wait...';
if isempty(h) % no handle given, create dialog
if usejavactrl&&usejava('swing') % Java swing implementation

% progress bar dialog
    h = progressfigure();

% progress bar
jbar = javax.swing.JProgressBar(0, 1000); % min--max
if ~isempty(r)
jbar.setValue(fix(1000*r));
jbar.setStringPainted(true);
else
    jbar.setStringPainted(false);
jbar.setIndeterminate(true);
end
pixelpos = getpixelposition(h);
    [jbarhgl, jbarctrl] = javacomponent(jbar, [0 0 pixelpos(3) pixelpos(4)], h);
set(jbarctrl, ...
'Tag', '__progressbarcontrol__', ... % used to differentiate this container from other containers
'Units', 'pixels');

% status message
if ~isempty(msg)
defaultmsg = msg;
end
uicontrol(h, ...
'Style', 'text', ...
'Units', 'pixels', ...
BackgroundColor,’white’, ...
’FontSize’, 8, ...
’String’, defaultmsg, ...
’Visible’, ’off’);
if ~isempty(msg)
set(h, ’Visible’, ’on’);
end

% progress bar dialog callbacks
set(h, ...
’UserData’, jbar, ...
’ResizeFcn’, @progressfigure_onresize);
progressfigure_onresize(h); % force control re-arrangement
set(h, ’Visible’, ’on’);
else% default MatLab implementation
assert(~isempty(r), ...
’gui:waitdialog:InvalidOperation’, ...
’waitdialogMatLab implementation cannot be in indeterminate state.’);
if msgupdate
    h = waitbar(r, msg);
else
    h = waitbar(r);
end
end
else% handle given, update dialog
jbar = get(h, ’UserData’);
if ~isempty(jbar) && isjava(jbar) % progress bar created with Java
    if ~isempty(r)
        jbar.setValue(fix(1000*r));
        jbar.setStringPainted(true);
        jbar.setIndeterminate(false);
    else
        jbar.setStringPainted(false);
        jbar.setIndeterminate(true);
    end
    % update status message
    if msgupdate
        msgctrl = findobj(h, ’Type’, ’uicontrol’, ’Style’, ’text’);
if ~isempty(msgctrl) && isscalar(msgctrl)
    visible = get(msgctrl, 'Visible');
if ~isempty(msg) % update string if given
    set(msgctrl, ...
    'String', msg, ...
    'Visible', 'on');
else
    set(msgctrl, ...
    'Visible', 'off');
end
if strcmp(visible, get(msgctrl, 'Visible')) % message visibility changed
    progressfigure_onresize(h); % force control re-arrangement
end
end
end
else% progress bar created with MatLab implementation
assert(~isempty(r), ...
'gui:waitdialog:InvalidOperation', ...
'waitdialogMatLab implementation cannot be in indeterminate state.');
if msgupdate
    h = waitbar(r, h, msg);
else
    h = waitbar(r, h);
end
end
end
drawnow;

function h = progressfigure()
% A figure for progress bar and associated controls.

width = 400;
height = 80;
h = figure(...
'Units', 'pixels', ...
'Color', 'white', ...
'DockControls', 'off', ...
'IntegerHandle', 'off', ...
'MenuBar', 'none', ...
'Name', 'Operation in progress', ...)
'NextPlot', 'new', ...
'NumberTitle', 'off', ...
'Pointer', 'watch', ...
'Position', [0, 0, width, height], ...
'Tag', '__progressbar__', ... % used to differentiate this dialog from other dialogs
'Toolbar', 'none', ...
'Visible', 'off', ...
'CloseRequestFcn', @(source,event) processfigure_onclose(source));
movegui(h, 'center'); % center on screen

% Occurs when the user closes the progress bar window.
function processfigure_onclose(fig)

cleanup = onCleanup(@() delete(fig));
set(fig, 'Name', 'Termination in progress');
drawnow;

function progressfigure_onresize(fig, event) %#ok<INUSD>
% Fired when the user resizes the progress bar window.
%
% Input arguments:
% fig: % a graphics handle to the progress bar window

ctrlheight = 20; % desired height of progress bar control in pixels
padding = 20; % desired padding around left and right edge of progress
bar control in pixels
[figwidth,figheight] = position2size(get(fig, 'Position'));

jbarctrl = findobj(fig, 'Tag', '__progressbarcontrol__'); % a MatLab HG
container that encapsulates the Java control
msgctrl = findobj(fig, 'Type', 'uicontrol', 'Style', 'text');

if ~isempty(msgctrl) && iscalar(msgctrl) && strcmp('on', get(msgctrl, 'Visible')); % a message text is present
set(jbarctrl, 'Position', [padding, (figheight-2*ctrlheight)/2, figwidth-
2*padding, ctrlheight]);
set(msgctrl, 'Position', [padding, (figheight-2*ctrlheight)/2+ctrlheight, 
figwidth-2*padding, ctrlheight]);
else % no message text
    set(jbarctrl, 'Position', [padding, (figheight-ctrlheight)/2, figwidth-
                            2*padding, ctrlheight]);
end

% A progress dialog to notify the user of the status an ongoing operation.
% Unlikewaitbar, the progress dialog is automatically closed when the
% operation is interrupted by the user or an error occurs.
%
% See also: waitdialog, waitbar

classdef ProgressDialog<UIControl
properties (Dependent)
    Control;
end
properties
    % Completion status p with 1 >= p >= 0, or empty if indeterminate.
    FractionComplete = 0;
    % Status message shown to the user.
    StatusMessage = 'Please wait...';
    % Indeterminate completion status.
    % Indeterminate status is intended for operations whose completion
    % status is unknown or cannot be computed.
    Indeterminate = false;
    Implementation = 'java';
end
properties (Access = private)
    Dialog;
end
methods
    function obj = ProgressDialog(varargin)
        obj = obj@UIControl(varargin{:});
    end

    function obj = Instantiate(obj, parent) %#ok<INUSD>
        switch obj.Implementation
        case 'matlab'
            impl = false;
        case 'java'
            impl = true;
        end

end
end
obj.Dialog = progressbar([], obj.FractionComplete, obj.StatusMessage, impl);
end

function control = get.Control(obj)
control = obj.Dialog;
end

function set.FractionComplete(obj, x)
if ~isempty(x)
    validateattributes(x, {'numeric'}, {'nonnegative','real','scalar'});
end
obj.FractionComplete = x;
if obj.TestForUserInterrupt()
callererror('gui:ProgressDialog', 'Operation terminated by user.');
end
obj.UpdateDialog();
end

function set.StatusMessage(obj, message)
if ~isempty(message)
    validateattributes(message, {'char'}, {'nonempty','row'});
end
obj.StatusMessage = message;
if obj.TestForUserInterrupt()
callererror('gui:ProgressDialog', 'Operation terminated by user.');
end
obj.UpdateDialog();
end

function set.Indeterminate(obj, tf)
validateattributes(tf, {'logical'}, {'scalar'});
obj.Indeterminate = tf;
obj.UpdateDialog();
end

function delete(obj)
if ishandle(obj.Dialog)
delete(obj.Dialog);
end
function uwait(obj)
    uwait(obj.Dialog);
end
end

methods (Access = private)
function tf = TestForUserInterrupt(obj)
    if ~ishandle(obj.Dialog)  % dialog has been closed by user
        obj.Dialog = [];
        tf = true;
    else
        tf = false;
    end
end
end

function UpdateDialog(obj)
    if obj.Indeterminate
        r = [];
    else
        r = obj.FractionComplete;
    end
    progressbar(obj.Dialog, r, obj.StatusMessage);
end
end
end