CHANNEL-STATE-ASSISTED CONVOLUTIONAL DECODING FOR SINGLE-INPUT SINGLE-OUTPUT IEEE 802.11N SYSTEMS

Author

Martha Vasiliki Sourla

Thesis Committee

Prof. Karystinos Georgios (Supervisor)

Prof. Liavas Athanasios

Prof. Christopoulos Dionysios

A thesis submitted in fulfillment

of the requirements for the degree of

Diploma in Electrical and Computer Engineering

TECHNICAL UNIVERSITY OF CRETE

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Technical University of Crete

Chania, Greece

August 1, 2024

Abstract

This thesis is on the implementation and simulation of single-input single-output IEEE 802.11n links and the examination of channel-state-assisted convolutional decoding to improve system performance. We present the full chain of transformations of the signal at the transmitter, according to the IEEE 802.11n standard, and explain the role of each module. We also explain the corresponding receiver modules. At each step, we present the parameters of the code that is used to implement each module and describe their use. Then, we focus on single-input single-output systems that employ convolutional coding. We derive closed-form approximations for the log-likelihood ratio (LLR) of the coded bits at the receiver end and, taking into account the specific signal constellation determined by the IEEE 802.11n standard, we show that, for any modulation and coding rate of the standard, the sequence of LLRs that is given as input to the channel decoder can be viewed as a sequence of received samples of binary phase-shift keying symbols in additive-white-Gaussian-noise with varying, but known, bit amplitude. The amplitude variation is due to the different channel state over different subcarriers. Then, we implement a channel decoding algorithm that is tailored to the derived LLR expressions, simulate the system performance over IEEE TGn Model-B channels, and compare with plain LLR-based decoding that does not incorporate the varying channel state.

Acknowledgements

The completion of this thesis marks the end of my studies towards the Diploma degree in Electrical and Computer Engineering. It marks the end of a wonderful journey that provided me with a lot of helpful skills for both my academic career and also my everyday life.

I would like to express my deepest gratitude to my supervisor, Prof. Georgios Karystinos, for his invaluable guidance throughout shaping this thesis. I am also grateful to the rest of my dissertation committee: Prof. Athanasios Liavas and Prof. Dionysios Christopoulos, for their contributions to my academic development.

I extend my appreciation to the faculty of the Technical University of Crete and its staff for their inspiring work. Their presence let so many of us pursue our goals.

Lastly, I am deeply indebted to my family and friends for their unconditional love, encouragement, and unwavering belief in my abilities. Their support has been a constant source of strength and motivation throughout this journey.

Contents

Abstract i

Acknowledgements ii

1 Introduction 1

1.1 Motivation 1

1.2 Thesis Outline 2

2 IEEE 802.11n and System Setup 3

2.1 Background 3

2.2 Orthogonal Frequency Division Multiplexing (OFDM) 4

2.3 Physical Layer (PHY) and Setup 5

3 Analysis of IEEE 802.11n Transmitter & Receiver 8

3.1 Transmitter 8

3.1.1 Information bits generation 8

3.1.2 Scrambling 8

3.1.3 Encoding 10

3.1.4 Interleaving . . . 13
3.1.5 Constellation Mapping . . . 16
3.1.6 OFDM Modulation . . . 18
3.1.7 Example . . . 18
3.2 Signal Transmission . . . 19
3.2.1 Channel Generation . . . 19
3.2.2 Waveform Transmission . . . 21
3.3 Receiver . . . 21
3.3.1 Packet Detection . . . 21
3.3.2 Offset Estimation . . . 25
3.3.3 Channel Estimation . . . 29
3.3.4 Data Recovery . . . 30

4 Viterbi Decoding 39
4.1 Background . . . 39
4.1.1 Convolutional Codes . . . 39
4.1.2 State Representation . . . 40
4.2 Viterbi Algorithm . . . 40
4.2.1 Implementation in IEEE 802.11n . . . 41
4.3 Leveraging Channel Information . . . 42
4.3.1 Overview of the Algorithm . . . 43
4.3.2 Advantages of Channel-State-Assisted Decoding . . . 45
4.3.3 BPSK vs QAM decoder . . . 46

5 Mathematical Processing **48**

5.1 Log Likelihood Ratio 48

5.1.1 BPSK 49

5.1.2 16-QAM 52

5.2 Log Likelihood Ratio with Constant Noise Variance 54

5.2.1 BPSK 54

5.2.2 16-QAM 54

6 Comparison **59**

6.1 BPSK 59

6.2 BPSK under Modified Channel Conditions 60

6.3 16-QAM 62

6.4 Observations 63

7 Conclusion **64**

List of Tables

2.1 Modulation Coding Schemes 6

3.1 Block interleaver parameters 14

List of Figures

2.1 HT-mixed PPDU format. 7

3.1 Transmitter structure (single input - single output (SISO)). 8

3.2 Encoding procedure. 13

3.3 Example transmitter procedure. 19

3.4 Signal Transmission. 21

3.5 Receiver structure (SISO). 22

3.6 Packet detection algorithm implemented by MATLAB. 24

4.1 BPSK vs QAM decoder. 46

5.1 BPSK Constellation Mapping. 50

5.2 QAM Constellation Mapping 52

6.1 BER vs. SNR for BPSK Modulation. 60

6.2 BER vs. SNR for BPSK Modulation Mild Modification. 61

6.3 BER vs. SNR for BPSK Modulation Extreme Modification. 62

6.4 BER vs. SNR for 16-QAM Modulation. 63

Acronyms

AWGN additive white gaussian noise

BCC binary convolutional coding

BPSK binary phase shift keying

BER bit error rate

CFO carrier frequency offset

CSD cyclic shift delay

FFT fast fourier transform

HT high throughput

HT-LTF high throughput long training field

ICI inter-carrier interference

ISI inter-symbol interference

L-LTF legacy long training field

L-STF legacy short training field

LLR log likelihood ratio

LDPC low-density parity-check

MAC medium access control

MCS modulation coding scheme

MIMO multiple input multiple output

NLOS non-line-of-sight

OFDM orthogonal frequency-division multiplexing

PHY physical layer

PPDU physical layer protocol data unit

PSDU physical layer service data unit

QAM quadrature amplitude modulation

QPSK quaternary phase shift keying

SNR signal-to-noise ratio

SISO single input - single output

WLAN wireless local area networks

Introduction

1.1 Motivation

The demand for higher data rates in wireless communication systems has been steadily increasing with the proliferation of mobile devices and the growing reliance on wireless networks for various applications. As users expect faster and more reliable connectivity, there is a critical need for the development of more efficient decoding algorithms to meet these demands.

The IEEE 802.11n standard, commonly known as Wi-Fi, has become one of the most widely used wireless communication standards, offering higher throughput and increased coverage compared to previous standards. However, achieving the full potential of 802.11n requires advancements in decoding techniques, particularly in environments with challenging channel conditions.

Motivated by the need to enhance the performance of wireless communication systems, this thesis focuses on optimizing the decoding algorithm for the IEEE 802.11n standard, one of the most fundamental wireless local area networks (WLAN) standards. Specifically, we aim to address the limitations of traditional decoding approaches by leveraging channel-state information derived from the orthogonal frequency-division multiplexing (OFDM) technique to enhance the reliability and efficiency of data transmission.

By developing a channel-state-assisted convolutional decoding algorithm, we seek to unlock new opportunities for enhancing the overall performance of wireless networks. This research aims to contribute to the ongoing evolution of wireless com-

munication technologies by providing practical solutions to the challenges faced in real-world deployment scenarios.

1.2 Thesis Outline

In the following sections, we will delve deeper into the background and context of the research, outlining the objectives, methodology, and contributions of this thesis. Chapter 2 provides a theoretical background of the 802.11 standards, covering key concepts such as the OFDM technique, modulation techniques, convolutional encoding, and Viterbi decoding. Chapter 3 analyzes the transmitter and receiver structures in the IEEE 802.11n standard, examining their components and functionalities in detail, as implemented in MATLAB. Chapter 4 outlines the Viterbi decoding algorithm and the methodology used for constructing the channel-state assisted convolutional decoder. Chapter 5 describes the mathematical processing of the log likelihood ratios, which are the input of the decoder. Chapter 6 presents the simulation results obtained from testing the proposed decoder, including performance metrics and comparisons with plain Viterbi decoding algorithm. Chapter 7 concludes the thesis by summarizing the main findings and discussing their implications.

IEEE 802.11n and System Setup

2.1 Background

The IEEE 802.11 standards define the protocols for implementing WLAN communications. The standard was first introduced in 1997 and has undergone several revisions and updates to improve performance, security, and interoperability. Key versions include 802.11a, 802.11b, 802.11g, 802.11n, and the latest 802.11ax, each introducing new features and enhancements to address the growing demands for wireless communication.

The IEEE 802.11 standard defines a protocol stack that includes the physical layer (PHY) and medium access control (MAC) layer. The PHY layer is responsible for the transmission and reception of data over the wireless medium, including modulation, encoding, and decoding of signals. The MAC layer manages access to the wireless medium, handling frame formatting, error control, and addressing, ensuring efficient communication without collisions. The interaction between these layers is crucial for achieving reliable and high-performance wireless communication.

The IEEE 802.11n is an amendment to the IEEE 802.11 wireless networking standard which increased maximum data rates to 600 Mbps by using four spatial streams at a channel width of 40 MHz improving the PHY and MAC layer efficiency. The evolution from previous standards such as IEEE 802.11a/b/g incorporated multiple new features including:

- **multiple input multiple output (MIMO):** Enhances data throughput and range by using multiple transmit and receive antennas.

- **Channel Bonding:** Combines two adjacent 20 MHz channels to create a 40 MHz channel, doubling the maximum data rate.

- **Frame Aggregation:** Increases throughput by sending multiple data frames in a single transmission.

2.2 Orthogonal Frequency Division Multiplexing (OFDM)

The IEEE 802.11n standard employs OFDM to achieve high data rates. OFDM is a multi-carrier modulation technique that divides a high-rate data stream into multiple lower-rate streams, which are then transmitted over several orthogonal subcarriers. This approach decreases the impact of multipath fading and inter-symbol interference (ISI) by ensuring that each subcarrier experiences nearly flat fading, simplifying channel equalization. Here's an explanation of the key concepts:

- **Subcarriers:** OFDM divides the spectrum into subcarriers that are spaced apart at precise frequencies. Typically, 64 subcarriers for a 20 MHz channel. Of these, 4 are utilized as pilot subcarriers for synchronization, phase tracking and channel estimation. Additionally, 52 subcarriers are used for data transmission. The remaining 8 subcarriers are mainly used as guard carriers or null subcarriers against interference from adjacent channels or sub-channels, thus minimizing inter-carrier interference (ICI).

- **Orthogonality:** Ensures that subcarrier signals are orthogonal to each other, reducing interference. The total bandwidth is divided into multiple narrower subbands, each carrying a separate subcarrier signal. The subcarriers are or-

thogonal, meaning the integral of the product of any two different subcarriers over one symbol period is zero. Mathematically, this can be expressed as:

$$\int_0^T e^{j2\pi f_i t} \cdot e^{-j2\pi f_j t}\, dt = 0 \quad \text{for} \quad i \neq j$$

where f_i and f_j are the frequencies of the i-th and j-th subcarriers, respectively, and T is the OFDM symbol period. This leads to efficient spectrum usage because subcarriers can overlap in the frequency domain without interfering with each other.

- **fast fourier transform (FFT):** Transforms a signal from the time domain to the frequency domain. FFT is used to modulate and demodulate the data into and out of these subcarriers.

We should note that between OFDM symbols, a guard interval with a cyclic prefix is inserted to prevent ISI.

2.3 Physical Layer (PHY) and Setup

The IEEE 802.11n standard features 32 different modulation coding scheme (MCS), that define the modulation, the coding rate, and the number of spatial streams. The following table displays the features of the MCS that will be simulated in this thesis. Only one spatial stream is used, channel bandwidth is set to 20 MHz and Guard Interval (GI) duration is 800ns.

In 802.11n three different formats are supported: the Non-HT format, the HT-mixed format, and the HT-greenfield format. The following figure showcases the

MCS	Modulation	Coding Rate	Data rate(Mb/s)
0	BPSK	1/2	6.5
1	QPSK	1/2	13.0
2	QPSK	3/4	19.5
3	16-QAM	1/2	26.0
4	16-QAM	3/4	39.0
5	64-QAM	2/3	52.0
6	64-QAM	3/4	58.5
7	64-QAM	5/6	65.0

Table 2.1: Modulation Coding Schemes

structure of the HT-mixed physical layer protocol data unit (PPDU), which is the one used in this thesis.

The non-HT part of the preamble contains a Legacy Short Training Field (L-STF), a Legacy Long Training Field (L-LTF), and a Legacy Signal (L-SIG) field. The L-STF is used for Carrier Frequency Offset (CFO) estimation and Packet Detection. The L-LTF is used for fine CFO estimation and the L-SIG field conveys information about rate and length.

The HT portion of the preamble consists of the HT-SIG, HT-STF, and HT-LTF fields. The HT-SIG field contains the information needed for the HT packet formats to be interpreted, for instance, information about the MCS or the channel bandwidth. The HT-STF improves automatic gain control estimation in MIMO systems. The HT-LTF is used for channel estimation at the receiver. More details about the fields of the preamble can be found in Sections 19.3.9.3 and 19.3.9.4 of IEEE802.11n (2022).

Two different Forward Error Correction code encoders are supported in the IEEE 802.11n Standard, the binary convolutional coding (BCC) and the low-density parity-check (LDPC) code. In this thesis, only the BCC Encoder is discussed.

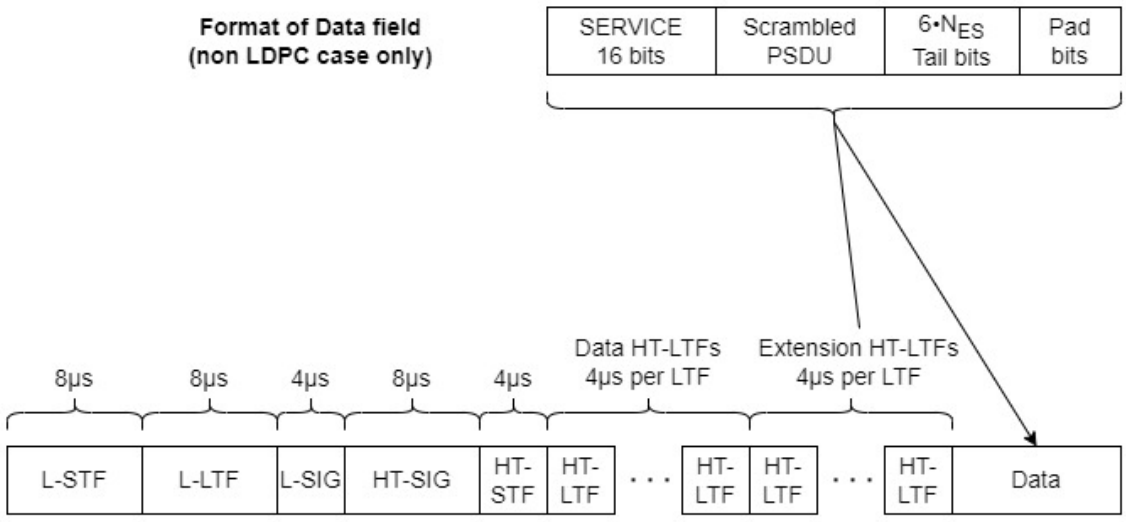

Figure 2.1: HT-mixed PPDU format.

The IEEE 802.11n Standard was the first to introduce MIMO support with up to 4 transmit and receive antennas. The a×b: c notation helps identify the capabilities of a system. The first number (a) is the number of transmit antennas, the second number (b) is the number of receive antennas and the third number (c) is the number of data spatial streams. In this thesis, a SISO 1x1 system with one spatial stream (1x1: 1) will be implemented.

Analysis of IEEE 802.11n Transmitter & Receiver

3.1 Transmitter

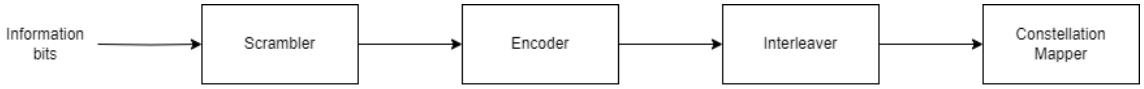

Figure 3.1: Transmitter structure (SISO).

3.1.1 Information bits generation

According to IEEE802.11n (2022), Section 19.3.11.2, 16 service bits are generated and set to 0. To simulate the transmission of the information bits, random bits (physical layer service data unit (PSDU)) are generated. Service bits and information bits are combined to simulate the transmitted signal denoted as DATA field in the standard.

3.1.2 Scrambling

A frame-synchronous scrambler is used to scramble and descramble the binary input signal. This function is commonly used in the transmitter and receiver of a Wi-Fi IEEE 802.11n system to add a certain level of randomness, reduce the likelihood of long sequences of identical bits and enhance the security and reliability of the transmitted data. Scrambling method is described in Section 3.2.1 in Logothetis (2021).

function y = wlanScramble(x, scramInit)

```
Input Parameters
```

- `x` : The input binary data that needs to be scrambled. It can be a binary column vector or a matrix of type 'int8' or 'double'.
- `scramInit` : The initial state of the scrambler. It can be an integer between 1 and 127 (inclusive), or a corresponding 7-by-1 column vector of binary bits of type 'int8' or 'double'.

`Output`

The function returns the scrambled (or descrambled) binary data as a binary column vector or matrix of the same size and data type as the input `x`. The output `y` contains the scrambled data.

`Function Overview`

The function first determines the data type of the input `x` and performs validation to ensure that the provided `x` is binary data and has two dimensions (column vector or matrix). It also validates the `scramInit` to ensure it meets the requirements specified in the IEEE802.11n (2022) standard.

After that, it generates a scrambling sequence of length 127 using the generator polynomial defined in IEEE802.11n (2022), Section 17.3.5.5, based on the provided `scramInit`. This scrambling sequence is used to add randomness to the input data.

Then the scrambling process takes place, in which the input data is divided into frames of size 127 (or less if the input is smaller). For each frame, the function performs XOR operations with specific bits of the scrambling sequence to scramble the data, according with the polynomial $S(x) = x^7 + x^4 + 1$. The scrambling sequence is updated for each frame, creating a frame-synchronous scrambling process. We should note here that each column of the input is scrambled independently with

the same initial state. The same scrambler structure is used for scrambling at the transmitter and descrambling at the receiver.

3.1.3 Encoding

Convolutional coding is a method of adding redundancy to the data before transmission to enhance the reliability of wireless communication. This redundancy enables the receiver to recover the original data even if some bits are corrupted during transmission. In IEEE802.11n (2022), specific guidelines are outlined in sections 17.3.5.6, 19.3.11.4, and 19.3.11.6 for the BCC encoder used in the transmitter. Each input data bit is processed along with the previous bits in a shift register based on the generator polynomials. A specific function is used to convert a convolutional code polynomial representation to a trellis structure, as described below.

function y = wlanBCCEncode(x, rate)

`Input Parameters`

- `x` : The binary input data to be encoded. It can be a binary matrix of class 'int8' or 'double', where each column represents a separate stream to be encoded.
- `rate` : The coding rate specified as a scalar, character vector, or string scalar. It can be 1/2, 2/3, 3/4, or 5/6.

`Output`

The function returns the binary matrix y, which contains the binary convolutionally encoded bits of the input data x. The number of rows of y is equal to the result of dividing the number of rows of input x by the specified rate, rounded to the next integer. The number of columns of y is equal to the number of columns of x.

`Function Overview`

The function begins by calling the `poly2trellis` function, which returns the state transition diagram of the decoder based on the specified parameters. According to IEEE802.11n (2022), the parameters are a constraint length (the number of previous input bits that affects the generation of each output bit), which is set to 7, and code generator vector [133 171] representing the octal numbers of the generator polynomials.

Subsequently, the function performs various validations and assignments. Additionally, it utilizes puncturing patterns, if necessary, to achieve different rates of convolutional encoding. These puncturing patterns are predefined according to the IEEE802.11n (2022) standard, the sections referenced earlier.

The encoding process is performed for each encoded stream in x using the `convenc function`. This function initializes the state of the trellis to zero and, following certain validations, it calls a built-in function (`convcore`) specifically designed for encoding.

function trellis = poly2trellis(constraintLength, codeGenerator)

`Input Parameters`

- `constraintLength` : A $1 \times k$ vector specifying the number of delays for each of the k input bit streams. This determines the "memory" or "order" of the convolutional encoder.

- `codeGenerator` : A $k \times n$ matrix of octal numbers: Specifies the n output connections for each of the k inputs. These numbers represent the coefficients of the polynomials used in the convolutional encoder.

`Output`

The output `trellis` is a structure containing the following fields:

- `numInputSymbols`: The number of input symbols, which is 2^k for a rate k/n code.
- `numOutputSymbols`: The number of output symbols, which is 2^n for a rate k/n code.
- `numStates`: The number of states in the trellis.
- `nextStates`: A matrix representing the next state transition from the current state for each possible input symbol. The rows represent the states and the columns represent the input bits.
- `outputs`: A matrix representing the output generated for each possible input symbol transition. Again the rows represent the states and the columns represent the input bits.

`Function Overview`

This function is responsible for converting the polynomial representation of the convolutional encoding to a trellis structure. As mentioned earlier, the call of this function is poly2trellis(7, [133 171]). The dimensions of the parameters indicate that, in the absence of a puncturing pattern, the encoding rate is $\frac{1}{2}$. The structure of the encoding process is shown in the figure below. It's important to note that the calculation of each output bit involves 7 bits: 6 previous bits and 1 input bit. For

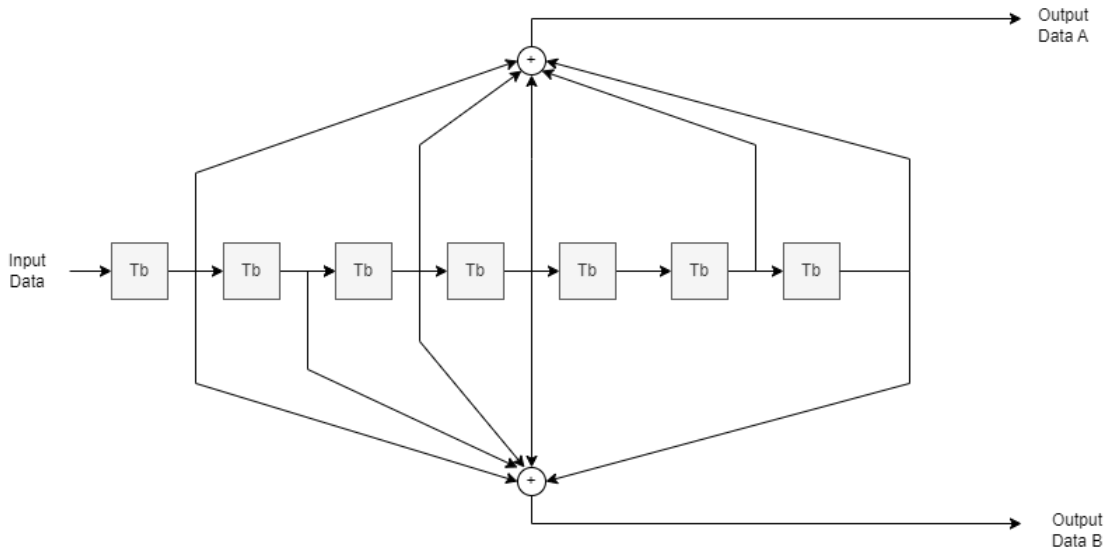

Figure 3.2: Encoding procedure.

the first output bit, labeled as A, the polynomial $133_8 = 1011011_2$ is used, while for the second output bit, B, the polynomial $171_8 = 1111001_2$ is utilized.

3.1.4 Interleaving

Interleaving is a technique used to enhance the robustness of data transmission. By rearranging the order of transmitted symbols, it minimizes the impact of burst errors and fading effects, which are common in wireless channels. The permutations that take place are described in the Sections 17.3.5.7 and 19.3.11.8 of **?** standard. These permutations help to shuffle the data bits creating a more uniform distribution of the errors. The first permutation maps adjacent coded bits onto non-adjacent subcarriers.

$$i = N_{\mathrm{ROW}}(k \bmod N_{\mathrm{COL}}) + \left\lfloor \frac{k}{N_{\mathrm{COL}}} \right\rfloor, \qquad k = 0, ..., N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - 1. \tag{3.1}$$

The second permutation maps adjacent coded bits alternately onto less and more significant bits of the constellation, to avoid long runs of low-reliability bits.

$$j = s(i_{\mathrm{SS}}) \cdot \left\lfloor \frac{i_{\mathrm{SS}}}{s(i_{\mathrm{SS}})} \right\rfloor + \left(i + N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - \left\lfloor N_{\mathrm{COL}} \frac{i}{N_{\mathrm{CBPSS}}(i_{\mathrm{SS}})} \right\rfloor \bmod s(i_{\mathrm{SS}}) \right), \quad i = 0, ..., N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - 1. \tag{3.2}$$

If there is more than one spatial stream, the third operation is applied and performs a frequency rotation to the additional spatial streams.

$$r = \left(j - \left((2(i_{\mathrm{SS}} - 1)) \bmod 3 + 3 \left\lfloor \frac{i_{\mathrm{SS}} - 1}{3} \right\rfloor \cdot N_{\mathrm{ROT}} \cdot N_{\mathrm{BPSCS}}(i_{\mathrm{SS}}) \right) \right) \bmod N_{\mathrm{CBPSCS}}(i_{\mathrm{SS}}), \quad j = 0, ..., N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - 1. \tag{3.3}$$

In this thesis, only MCS with one spatial stream are used, so the third permutation is ignored.

Parameter	20MHz
$\mathrm{N_{COL}}$	13
$\mathrm{N_{ROW}}$	$4 \cdot \mathrm{N_{BPSCS}(i_{SS})}$
$\mathrm{N_{ROT}}$	11

Table 3.1: Block interleaver parameters

function y = wlanBCCInterleave(x, type, numCBPSSI)

`Input Parameters`

- `x` : The input binary data to be interleaved. It should be a 3D array of type 'int8' or 'double' with dimensions (Ncbpssi * Nsym) x Nss x Nseg. Ncbpssi is the number of coded bits per OFDM symbol per spatial stream per interleaver block, Nsym is the number of OFDM symbols, Nss is the number of spatial streams, and Nseg is the number of segments.

- `type` : A character vector or string specifying the type of interleaving to be performed. It can be either 'Non-HT' or 'VHT', in our case is set to 'VHT'.

- `numCBPSSI` : The number of coded bits per OFDM symbol per spatial stream per interleaver block. It is a positive integer scalar and depends on the specific standard used. In our case, it is set to 52, like the number of data subcarriers.

- `chanBW` : A character vector or string with the channel bandwidth. It must be one of: 'CBW1', 'CBW2', 'CBW4', 'CBW8', 'CBW16', 'CBW20', 'CBW40', 'CBW80', or 'CBW160' where the integer following CBW denotes the bandwidth in MHz. This parameter is optional because in 'Non-HT' interleaver type it is not required.

`Output`

The output variable y is an (Ncbpssi * Nsym) x Nss x Nseg array of the same class as the input x, containing the binary convolutionally interleaved data. (Nss and Nseg are equal to 1).

`Function Overview`

At first, the function validates the input arguments and determines the channel bandwidth (chanBW) and the number of coded bits per subcarrier per spatial stream (numBPSCS) based on the interleaver type.

The function then calculates the interleaving parameters (Ncol, Nrow, Nrot) based on the type, numCBPSSI, numBPSCS, and chanBW. It performs binary convolutional interleaving on the input data using the calculated parameters with the function `bccInterleaveCore`, which applies the three interleaving permutations stated in the standard.

In our case, type is set to 'VHT' and numCBPSSI is 52, like the number of data subcarriers. Inside `wlanBCCInterleave`, the internal function `bccInterleaveCore` is called, which applies the three interleaving permutations stated in Section 17.3.5.7 of IEEE802.11n (2022) standard.

3.1.5 Constellation Mapping

Constellation mapping is called the procedure of mapping the interleaved bits to constellation points (complex symbols) based on the MCS. The modulations that IEEE 802.11 standard support are binary phase shift keying (BPSK), quaternary phase shift keying (QPSK) and quadrature amplitude modulation (QAM), specifically 16-QAM and 64-QAM.

function y = wlanConstellationMap(x, numBPSCS)

`Input Parameters`

- `x` : x is a binary 'int8' or 'double' vector, matrix, or multidimensional array containing the input bits to map into symbols. The number of rows of X must

be an integer multiple of NUMBPSCS.

- `numBPSCS` : The number of coded bits per subcarrier per spatial stream. It is an integer, whose values can be 1 (BPSK), 2 (QPSK), 4 (16-QAM), 6 (64-QAM) and is equal to $\log_2(M)$, where M is the modulation order.

`Output`

The output `y` is a complex vector, matrix, or multidimensional array containing the mapped symbols. It has the same size as the input `x` except for the number of rows, which is equal to the number of rows of `x` divided by `numBPSCS`.

`Function Overview`

The function performs constellation mapping, after doing some validations for the inputs, depending on the modulation scheme (`numBPSCS`). For BPSK modulation (n`numBPSCS`=1), the constellation is [-1, 1], and the binary input bits are mapped to the corresponding complex symbols ($0 \rightarrow -1$ and $1 \rightarrow 1$) . For other modulation schemes (`numBPSCS`>1), the function uses 'comm.internal.wlanQAMSymbolMap' to get the symbol mapping. The symbol mapping is a vector containing the decimal represantation of the modulation scheme as shown in the figure 17-10 Section 17.3.5.8, IEEE802.11n (2022). It utilizes MATLAB's 'comm.internal.qam.modulate' function to perform constellation mapping of the input `x` into complex symbols. If the input `x` has more than three dimensions, the function reshapes it into a column to support the 'comm.internal.qam.modulate' function, and then reshapes it back to the original size after the mapping. If the phase argument is provided as a third input, the function rotates the constellation points counter-clockwise by the specified amount in radians. The mapping is performed column-wise for the input `x`.

3.1.6 OFDM Modulation

OFDM modulation is performed by MATLAB's **wlanOFDMModulate**, which is a function that performs OFDM modulation on the frequency domain signal and returns the time-domain signal.

function y = wlanOFDMModulate(x,cplen,varargin)

`Input Parameters`

- `x`: X is the Nc-by-Nsym-by-Nt frequency-domain signal, where Nc represents the number of frequency subcarriers, Nsym represents the number of OFDM symbols, and Nt represents the number of antennas in the spatial-domain.
- `cplen`: A non-negative, integer scalar representing the cyclic prefix length for all OFDM symbols.

`Output`

The output y is the modulated time-domain signal.

`Function Overview`

The function performs OFDM modulation, by applying the inverse Fourier transform to the input signal.

3.1.7 Example

The above example represents a SISO system with bandwidth 20 MHz. The modulation used is 16-QAM (cfgHT.MCS = 4) with coding rate 3/4. To illustrate the information bits, 8000 random bit are generated in PSDU. After that, service, tail and pad bits are added according to Section 17.3.5 of IEEE802.11n (2022). Data is

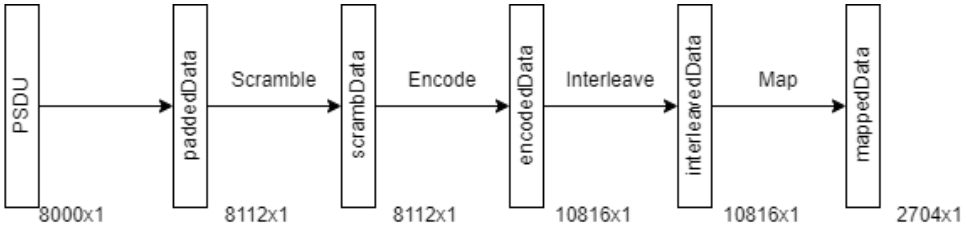

Figure 3.3: Example transmitter procedure.

then scambled and encoded with a rate of 3/4 and the number of rows is the result of 8112/(3/4). Consequently, the encoded data is interleaved as described above and then, the interleaved is mapped according to the modulation. The number or rows is 10816/4 where 4 represents the number of coded bits per subcarrier per spatial stream.

3.2 Signal Transmission

The channel generation process emulates the wireless communication channel to replicate real-world wireless propagation conditions.

3.2.1 Channel Generation

First, a HT format configuration object, that contains the transmit parameters for the HT-Mixed Format, has to be created. This is accomplished by calling MATLAB's **wlanHTConfig** function.
cfgHT = wlanHTConfig; The following properties are set:

- The channel bandwidth is set to 20 MHz.
- The number of transmit antennas is set to the desired number.

- The number of space-time streams is set to the desired number.
- PSDU length in bytes, specified as an integer in the interval [0, 216 – 1].
- MCS is set to the desired MCS.
- Channel Coding method is set to BCC.
- Spatial Mapping Method is set to Fourier.

The next step is the configuration of the channel. IEEE TGn multipath fading channel is used, especially Model - B, Schumacher (2004). Distance between transmitter and receiver is set to 10 m., so that the model is non-line-of-sight (NLOS). For this purpose MATLAB's **wlanTGnChannel** is utilized and it creates a system object for the TGn Channel, as specified by the IEEE 802.11 WLAN Working group. The following parameters are set:

tgnChannel = wlanTGnChannel;

tgnChannel.DelayProfile = 'Model-B';

tgnChannel.NumTransmitAntennas = cfgHT.NumTransmitAntennas;

tgnChannel.NumReceiveAntennas = NumRxAnts;

tgnChannel.TransmitReceiveDistance = 10;

tgnChannel.LargeScaleFadingEffect = 'None';

tgnChannel.RandomStream = 'mt19937ar with seed';

tgnChannel.Seed = double(seeds(channelIndex,1));

tgnChannel.PathGainsOutputPort = 1;

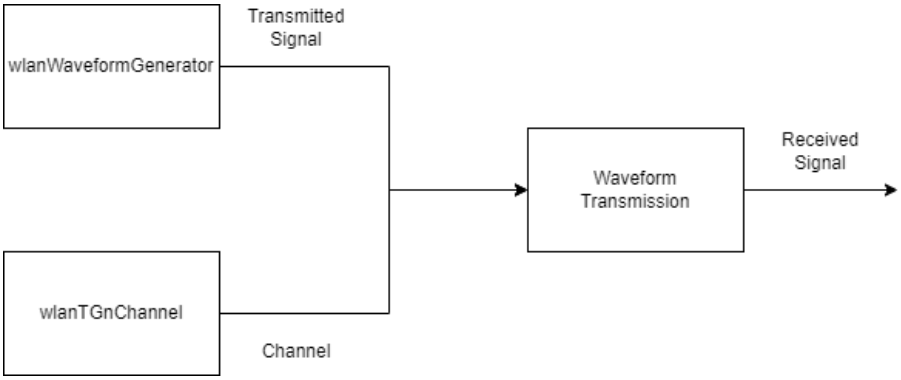

Figure 3.4: Signal Transmission.

3.2.2 Waveform Transmission

The generated waveform is passed through the TGn channel model (wlanTGnChannel) to simulate the effects of the wireless channel, including path loss, multipath fading, and noise. It is important to note that in this case the received signal is noiseless. Noise can be added after the signal is received. It is important to recognize that the utilization of wlanTGnChannel yields a channel that exhibits temporal variation, a characteristic that can introduce complications in the simulation process. In order to avoid that, we use the first path gain generated by wlanTGnChannel across all time instances. Subsequently, the waveform is propagated through this static channel.

3.3 Receiver

3.3.1 Packet Detection

Packet detection is the initial step in the reception process, where the presence of a valid packet is identified within the received signal. According to the IEE802.11

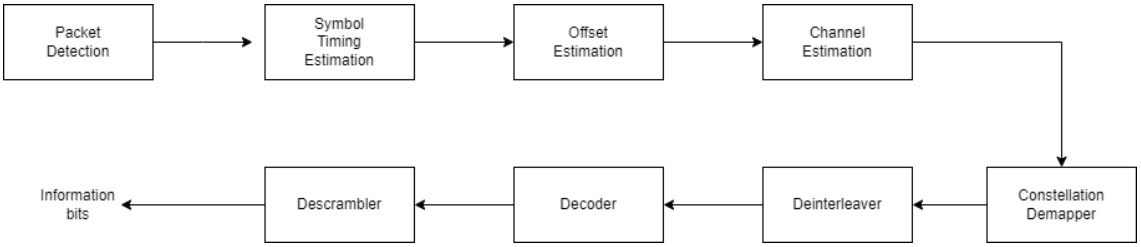

Figure 3.5: Receiver structure (SISO).

standard, preamble detection is used for this purpose. Specifically, the legacy short training field (L-STF) segment of the preamble is used to compute the offset from the beginning of the input waveform to the detected preamble's start using auto-correlation.

function startoffset= wlanPacketDetect(x, chanBW)

`Input Parameters`

- `x`: A single or double Ns-by-Nr matrix of real or complex samples, representing the received time-domain signal. Ns stands for the number of time domain samples and Nr for the number of receive antennas. In SISO systems, Nr is equal to 1.

- `chanBW`: A character vector or string describing the channel bandwidth. Its value can be 'CBW5', 'CBW10', 'CBW20', 'CBW40', 'CBW80', 'CBW160', or 'CBW320'. For our system 'CBW20' is used.

`Output`

The function returns an integer scalar indicating the location of the start of a detected packet as the offset from the start of the matrix X. If no packet is detected an empty value is returned.

`Function Overview`

At first, validations are performed and the length of the L-STF symbol is computed. Afterwards zeros are added to the input signal, to make its length equal to half of the length of the L-STF. The input waveform is then separated in blocks of half of the L-STF length, so that it can be processed gradually. Packet detection is accomplished with the use of the function `correlateSamples`, which is also defined in wlanPacketDetect.m file.A loop is used, so that the input rxSig is each time the block of the original input signal X that we are processing.

function [packetStart,Mn] = correlateSamples(rxSig, symbolLength, lenLSTF, threshold)

`Function Overview`

The received signal, rxSig, is delayed and then correlated in two sliding windows independently. The packet detection processing output provides decision statistics (m_n) of the received waveform. The following image, provided by MATLAB's Documentation for `wlanPacketDetect`, demonstrates how the function `correlateSamples` works.

Window C auto-correlates between the received signal and the delayed version, C_n.

$$C_n = \sum_{l=1}^{Nr} \sum_{K=0}^{D-1} r_{n+k,l} r^*_{n+k+D,l} \tag{3.4}$$

D is the period of the L-STF short training symbols.

Window P calculates the energy received in the auto-correlation window, P_n.

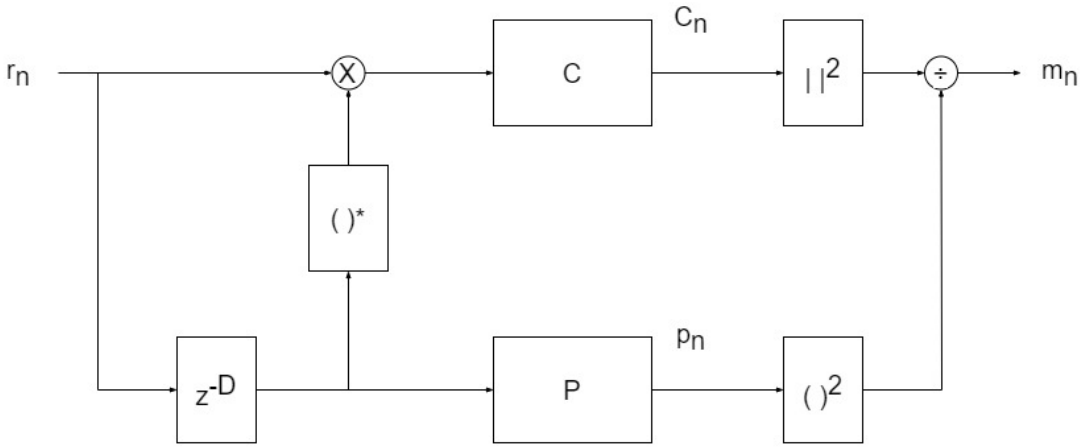

Figure 3.6: Packet detection algorithm implemented by MATLAB.

$$P_n = \sum_{l=1}^{Nr} \sum_{K=0}^{D-1} |r_{n+k+D,l}|^2 \tag{3.5}$$

The decision statistics, M_n, normalize the auto-correlation by P_n so that the decision statistic is not dependent on the absolute received power level.

$$m_n = \frac{|c_n|^2}{{p_n}^2} \tag{3.6}$$

The values of M_n that are not greater than the selected threshold, which in our case is the default threshold that is equal to 0.5, are discarded. Lastly, the relative distance between the first peak and subsequent peaks is checked, so that it does not exceed three times the symbol length. If it does, then no packet is detected. Each peak corresponds to the detection of a single packet. The output argument packetStart of the function `correlateSamples` is also the output argument startoffset of the function `wlanPacketDetect`.

It should be mentioned that the algorithm presented above is based on the fact that the L-STF sequences have good correlation properties and a low peak-to-average power, as stated in Section 3.1.3. of Logothetis (2021).

3.3.2 Offset Estimation

For accurate data recovery, synchronization between the transmitter and the receiver is essential. In IEEE 802.11 standard, offset estimation is responsible for aligning the received signal with the timing reference established by the transmitter. A correlation-based method and preamble fields are used for these timing offsets, and several functions are employed to achieve this. Firstly, the `wlanCoarseCFOEstimate` performs coarse carrier frequency offset (CFO) estimation using the L-STF of the preamble, as detailed in van Zelst and Schenk (2004). Following coarse CFO estimation, the `helperFrequencyOffset` function, a modified version of MATLAB's `FrequencyOffset` which is adjusted so that it can also be used for MIMO systems. For our purposes, these adjustments can be disregarded. Subsequently, the `wlanSymbolTimingEstimate` is utilized for the fine symbol timing estimation with the help of legacy long training field (L-LTF) field of the preamble. Finally, fine CFO estimation is performed using `wlanFineCFOEstimate` function after symbol timing estimation.

function foffset = wlanCoarseCFOEstimate(in,chanBW)

`Input Parameters`

- `in`: A single or double complex matrix of size Ns-by-Nr. Ns is the number of time domain samples in the L-STF, and Nr is the number of receive antennas.

- `chanBW`: A character vector or string describing the channel bandwidth. Its value can be 'CBW5', 'CBW10', 'CBW20', 'CBW40', 'CBW80', 'CBW160', or 'CBW320'. For our system 'CBW20' is used.

`Output`

The function returns a double scalar representing the carrier frequency offset in Hertz.

`Function Overview`

After several validations are performed, the number of samples inside the L-STF is calculated. Samples are then extracted and the parameters for the estimation are set, assuming 4 repetitions per FFT period. Internal function `cfoEstimate` is called. CFO estimation is then performed and CFO is computed based on the phase of delay correlation of weighted signals.

function out = helperFrequencyOffset(in, fs, foffset)

`Input Parameters`

- **in**: A complex 2D array.
- **fs**: A double scalar representing the sampling rate in Hertz.
- **foffset**: A double scalar containing the frequency offset to be applied to the input in Hertz.

`Output`

The function returns the frequency-offset output, which is the same size as 'in'.

`Function Overview`

The function creates a vector t of time samples representing the time axis of the input signal. This time vector is used to modulate the input signal with a complex exponential, effectively applying the frequency offset. For each antenna (column) in the input signal, it applies the frequency offset correction using the complex exponential term $e^{i \cdot 2\pi \cdot f_{offset} \cdot t}$ which rotates the phase of each sample in the input signal by an amount proportional to the time index t and the frequency offset f_{offset}. This results in a frequency shift in the time domain, compensating for the carrier frequency offset.

function startOffset = wlanSymbolTimingEstimate(x, chanBW)

`Input Parameters`

- `x`: A single or double matrix of size Ns-by-Nr, containing real or complex values. This is the received time-domain signal.

- `chanBW`: A character vector or string describing the channel bandwidth. Its value can be 'CBW5', 'CBW10', 'CBW20', 'CBW40', 'CBW80', 'CBW160', or 'CBW320'. For our system 'CBW20' is used.

`Output`

The function returns an integer within the range [-L, Ns-2L], where L denotes the length of the L-LTF.

`Function Overview`

After the proper validations, the threshold is set to 1 by default, if not given as an input parameter. The threshold is used to determine the decision metric's level, indicating a successful symbol timing estimate. Then the L-LTF is computed

based on the chanBW and the cross-correlation between x and L-LTF is computed. The decision metric M is computed based on the cross-correlation results. It measures the similarity between the received signal and the L-LTF, indicating potential symbol timing positions. Then, it takes into account the cyclic shift delay (CSD) and startOffset is returned which is the timing offset between the start of the input waveform and the estimated start of L-STF.

function foffset = wlanFineCFOEstimate(in,chanBW)

`Input Parameters`

- `in`: A single or double complex matrix of size Ns-by-Nr, containing real or complex values. Ns is the number of time domain samples in the L-LTF of the preamble.

- `chanBW`: A character vector or string describing the channel bandwidth. Its value can be 'CBW5', 'CBW10', 'CBW20', 'CBW40', 'CBW80', 'CBW160', or 'CBW320'. For our system 'CBW20' is used.

`Output`

The function returns a double scalar representing the carrier frequency offset in Hertz.

`Function Overview`

This function operates the same calculations as `wlanCoarseCFOEstimate`, however L-LTF is used, rather than L-STF. In this case also 2 repetitions per FFT period are assumed. Fine CFO has to be applied to the input signal with the use of `helperFrequencyOffset`.

3.3.3 Channel Estimation

Channel estimation allows the receiver to accurately estimate the channel characteristics so as to minimize the effects of channel impairments, such as fading and interference. To perform channel estimation, the demodulated high throughput long training field (HT-LTF) field is utilized, which is extracted from the received time-domain HT-LTF of the preamble with the help of `wlanHTLTFDemodulate` function. The demodulated signal is parsed as an input to the function `wlanHTLTFChannelEstimate` that performs channel estimation between all space-time, extension streams and receive antennas.

function y = wlanHTLTFDemodulate(rx, cfgHT)

`Input Parameters`

- `rx`: A complex matrix containing the received time-domain HT-LTF signal. It is of size Ns-by-Nr.
- `cfgHT`: A packet format configuration object for the WLAN high throughput (HT) packet format.

`Output`

The function returns a complex matrix or 3-D array of size Nst-by-Nsym-by-Nr, that represents the frequency-domain signal corresponding to the HT-LTF. Nst is the number of data and pilot subcarriers and Nsym is the number of OFDM symbols in the HT-LTF.

`Function Overview`

First, some validations are performed. Afterwards,the HT-LTF sequence, as well

as the OFDM configuration are obtained. The internal function `legacyOFDMDemodulate` is then called, which also calls the internal function `ofdmDemodulate`, that performs the demodulation by applying FFT to the input signal. Non-active subcarriers are then discarded and lastly the remaining signal is denormalized and phase rotation is removed from the subcarriers. The demodulated signal can now be returned.

function est = wlanHTLTFChannelEstimate(rxSym,cfgHT)

`Input Parameters`

- `x`: A complex array containing demodulated HT-LTF OFDM symbols. Array size is Nst-by-Nsym-by-Nr.
- `cfgHT`: A packet format configuration object for the WLAN HT packet format.

`Output`

The function returns a complex array containing the estimated channel at data and pilot subcarriers. The size of the array is Nst-by-(Nsts+Ness)-by-Nr.

`Function Overview`

Inside the function, several validations are made and then after getting the OFDM configuration the internal function `htltfEstimate` is called. For SISO systems, Least Squares (LS) estimation is used. This means that we get the channel estimate by dividing the received signal with the input signal, in our case the HT-LTF part of the preamble.

3.3.4 Data Recovery

The receiver recovers the transmitted data (PSDU) from the received HT-Data field using the channel estimate (chanEst) and noise power estimation (nVarHT). This is

accomplished with the usage of **helperWlanHTDataRecover** function, which is a modification of MATLAB's **WlanHTDataRecover**.

OFDM Demodulation

OFDM demodulation is performed by MATLAB's **wlanOFDMDemodulate**, which is a function that performs OFDM demodulation on the time-domain signal and returns the frequency domain signal, separating the data from the pilot subcarriers. **function [data, pilot] = wlanOFDMDemodulate(x, cfgOFDM, ofdmSymOffset)**

`Input Parameters`

- `x`: X is the data part of the time-domain received signal.
- `cfgOFDM`: cfgOFDM is a struct containing the size of the FFT and the length of the cyclic prefix.
- `ofdmSymOffset`: ofdmSymOffset is a scalar containing the OFDM symbol offset.

`Output`

The output arguments of the function are two matrices `data` and `pilot` containing the information of the data and pilot subcarriers respectively.

`Function Overview` The function performs OFDM demodulation, by removing the cyclic prefix and applying the Fourier transform to the input signal.

Equalizing

Equalization is a technique used to reverse the distortion that is incurred when a signal is transmitted through a channel. This is performed by MATLAB's internal function

function [y, CSI] = wlanEqualize(x, chanEst, eqMethod, varargin)

`Input Parameters`

- `x`: x is the input signal and is of size Nsd x Nsym x Nr, where Nsd represents the number of data subcarriers (frequency domain), Nsym represents the number of OFDM symbols (time domain), and Nr represents the number of receive antennas (spatial domain).
- `chanEst`: chanEst is the channel estimate and is of size Nsd x Nsts x Nr, where Nsts represents the number of space-time streams.
- `eqMethod`: eqMethod is the specified equalization method. In this thesis, the equalization method is set to MMSE.
- `varargin` : Additional optional parameters:
  - `noiseVar`: noiseVar is a single or double-precision, real, nonnegative scalar representing the noise variance.

`Output` The single or double precision output `y` is of size Nsd x Nsym x Nsts and it contains the estimate of the transmitted signal. `y` is complex when either X or

CHANEST is complex and is real otherwise. The single or double precision, real output CSI is of size Nsd x Nsts and contains the channel state information. `Function Overview` For a SISO system, the function performs equalization by dividing the input signal by the norm of the channel and multiplying it by the conjugate of the channel estimate.

Constellation Demapping

Constellation Demapping is the process of converting the received equalized complex symbols back to interleaved bits or log likelihood ratio (LLR) based on the MCS. This function, **wlanConstellationDemap**, supports both hard-decision and soft-decision demodulation. Hard-decision demodulation directly maps the received symbols to the nearest constellation points, resulting in bit values. Soft-decision demodulation, on the other hand, computes LLRs, which are real numbers that provide information about the confidence of each bit's value, enhancing error correction performance. **function y = wlanConstellationDemap(x, noiseVar, numBPSCS, varargin)**

`Input Parameters`

- `x` : The received symbols. It should be a single or double precision vector, matrix, or multidimensional array containing the received symbols.
- `noiseVar` : A single or double nonnegative scalar representing the noise variance estimate. For 'hard' demapping, this is ignored.
- `numBPSCS` : A scalar specifying the number of coded bits per subcarrier per

spatial stream. It can be 1 (BPSK), 2 (QPSK), 4 (16-QAM), 6 (64-QAM) and is equal to $\log_2(M)$, where M is the modulation order.

- `varargin` : Additional optional parameters:
  - `DEMAPTYPE` : A character vector or string scalar specifying the demapping type. It can be 'hard' for hard-decision demapping or 'soft' for soft-decision approximate LLR method. The default is 'soft'.
  - `PHASE` : A scalar, matrix, or multidimensional array specifying the phase rotation in radians. The phase and mapped symbols must have compatible sizes.

`Output`

The output `y` is a matrix or multidimensional array containing the demapped symbols. It has the same data type as the input `x` and the same size except for the number of rows, which will be equal to the number of rows of `x` multiplied by `numBPSCS`.

`Function Overview`

The function performs constellation demapping of the input symbols `x`. It supports both hard-decision demapping, which maps the received symbols to the nearest constellation points to produce bit values, and soft-decision demapping, which computes LLRs. The noise variance `noiseVar` is used in the LLR computation for 'soft' demapping. LLRs are real numbers that represent the confidence of each bit's value, where a positive LLR indicates that the bit is more likely to be 0, with the magnitude indicating the certainty of this decision.

Deinterleaving

Deinterleaving is the process of inversing the three permutations that were permormed in the interleaving process. The inverse of the third permutation is

$$j = \Big(r + \Big((2(i_{\mathrm{SS}} - 1)) \bmod 3 + 3\Big\lfloor \frac{i_{\mathrm{SS}} - 1}{3} \Big\rfloor \cdot N_{\mathrm{ROT}} \cdot N_{\mathrm{BPSCS}}(i_{\mathrm{SS}})\Big)\Big) \bmod N_{\mathrm{CBPSCS}}(i_{\mathrm{SS}}),$$
$$r = 0, 1, ..., N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - 1. \tag{3.7}$$

The inverse of the second permutation is

$$i = s(i_{\mathrm{SS}}) \cdot \Big\lfloor \frac{j}{s(i_{\mathrm{SS}})} \Big\rfloor + \Big(j + \Big\lfloor N_{\mathrm{COL}} \frac{j}{N_{\mathrm{CBPSS}}(i_{\mathrm{SS}})} \Big\rfloor \bmod s(i_{\mathrm{SS}})\Big), \qquad j = 0, 1, ..., N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - 1. \tag{3.8}$$

The inverse of the first permutation is

$$k = N_{\mathrm{COL}} \cdot i - (N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - 1) \cdot \Big\lfloor \frac{i}{N_{\mathrm{ROW}}} \Big\rfloor, \qquad i = 0, 1, ..., N_{\mathrm{CBPSS}}(i_{\mathrm{SS}}) - 1. \tag{3.9}$$

In MATLAB this process is executed by

function y = wlanBCCDeinterleave(x, type, numCBPSSI, varargin)

`Input Parameters`

- `x`: A single or double-precision array of size (Ncbpssi*Nsym)-by-Nss-by-Nseg containing binary convolutional interleaved data.

- `type`: A character vector or string specifying the type of deinterleaving to perform. It must be 'Non-HT' or 'VHT'.

- `numCBPSSI`: A positive integer scalar containing the number of coded bits per OFDM symbol per spatial stream per interleaver block. It is equal to Nsd*Nbpscs for 'Non-HT' TYPE, and equal to Nsd*Nbpscs/Nseg for 'VHT' TYPE, where Nsd is the number of data subcarriers, and Nbpscs is the number of coded bits per subcarrier per spatial stream.

- `varargin` : Additional optional parameters:

  - `CHANBW`: A character vector or string with the channel bandwidth. It must be one of 'CBW1', 'CBW2', 'CBW4', 'CBW8', 'CBW16', 'CBW20', 'CBW40', 'CBW80', or 'CBW160'. CHANBW is not a required parameter for the 'Non-HT' deinterleaver TYPE.

`Output`

The output `y` is an array of size (Ncbpssi*Nsym)-by-Nss-by-Nseg containing binary convolutionally deinterleaved data. Ncbpssi is the number of coded bits per OFDM symbol per spatial stream per interleaver block, Nsym is the number of OFDM symbols, Nss is the number of spatial streams, and Nseg is the number of segments.

`Function Overview`

The function outputs binary convolutionally deinterleaved data for the specified interleaver type.

Decoding

To retrieve the decoded symbols encoded using a binary convolutional code, the function **wlanBCCDecode** is utilized which supports both hard-decision and soft-decision Viterbi (1998) algorithms. Hard-decision decoding directly maps the received symbols to bit values, whereas soft-decision decoding computes LLRs to provide more nuanced information about each bit's confidence.

function y = wlanBCCDecode(x, rate, varargin)

`Input Parameters`

- `x` : The input symbols to decode. It should be a matrix of type single, double, or int8, containing encoded symbols. For 'soft' decoding, `x` should be a real matrix of log-likelihood ratios, where positive values represent logical 0 and negative values represent logical 1.

- `rate` : A scalar, character vector, or string scalar specifying the coding rate. Possible values are 1/2, 2/3, 3/4, or 5/6, or their corresponding string representations '1/2', '2/3', '3/4', or '5/6'.

- `varargin` : Additional optional parameters:

  - `DECTYPE` : A character vector or string scalar specifying the decoding type. It can be 'hard' for hard-decision decoding or 'soft' for soft-decision decoding without quantization. The default is 'soft'.

  - `TDEPTH` : A positive integer scalar specifying the traceback depth of the Viterbi decoding algorithm.

`Output`

The output `y` is a binary int8 matrix containing the decoded bits. The number of rows of `y` is equal to the number of rows of input `x` multiplied by the coding rate, rounded to the next integer. The number of columns of `y` is equal to the number of columns of `x`.

`Function Overview`

The function performs binary convolutional decoding of the input symbols `x`. It supports both hard-decision decoding, which maps the received symbols to bit values, and soft-decision decoding, which computes LLRs. The optional traceback depth parameter `TDEPTH` controls the depth of the Viterbi algorithm's traceback, affecting decoding performance and complexity.

Descrambling

Descrambling is the process of reversing the scrambling applied to the transmitted data, restoring the original binary sequence. The `wlanScramble` function is used for both scrambling and descrambling, as it implements a frame-synchronous scrambler described in IEEE802.11n (2022) standard. The initial state must match the state used during scrambling, otherwise if the initial state `scramInit` is all zeros, descrambling is bypassed, and the data is returned without modification.

Viterbi Decoding

4.1 Background

The IEEE 802.11n standard incorporates convolutional coding to enhance data integrity and facilitate error correction in wireless communications. Convolutional codes introduce redundancy into the data stream, allowing receivers to detect and correct errors during transmission. The Viterbi algorithm, developed by Andrew Viterbi in 1967, is a dynamic programming algorithm that provides a means for efficient maximum likelihood decoding of these convolutional codes.

4.1.1 Convolutional Codes

Convolutional codes apply a sequence of XOR operations over the input data stream, generating multiple output bits for each input bit. These output bits are sent over the communication channel, introducing redundancy that can help in error correction on the receiver side. The following equation represents the convolutional encoding process:

$$Y_i = \sum_{j=0}^{k} G_{ij} \cdot X_{i-j} \tag{4.1}$$

where Y_i are the output bits, G_{ij} are the generator polynomials, and X_i are the input bits.

4.1.2 State Representation

The decoding process involves navigating through a trellis diagram, where each state represents a potential sequence of encoded bits. Each transition between states corresponds to a possible transmitted bit, and the Viterbi algorithm systematically evaluates paths through this trellis to determine the most likely transmitted sequence.

4.2 Viterbi Algorithm

The Viterbi algorithm operates by maintaining all possible paths through the trellis but selects and extends only the most probable path at each stage. Here's a detailed step-by-step explanation of the Viterbi decoding process:

Initialization

The algorithm starts by initializing the path metrics. Initially, all path metrics are set to infinity, except for the starting state, which is set to zero. This setup accounts for the known initial state of the convolutional encoder.

Then, the algorithm proceeds through multiple steps to reach all possible states from the initial state. This step count may vary based on the specific system configuration and requirements. After completing these steps, the algorithm moves into the recursion phase, where it continuously updates the path metrics based on the received data and the transition probabilities.

Recursion

During the recursion step, the algorithm calculates the path metric for each possible state at each time step, based on the metrics of previous states and the received symbols. The path metric update equation is given by

$$M_t(s) = \min\left[M_{t-1}(s') + d(y_t, x_t)\right] \tag{4.2}$$

where $M_t(s)$ is the path metric for state s at time t, s' is a previous state, and $d(y_t, x_t)$ is the branch metric or distance measure between the received symbol y_t and the expected symbol x_t.

Traceback

Once the recursion is complete, the algorithm traces back from the state with the minimum metric in the final time step, reconstructing the most likely transmitted sequence.

4.2.1 Implementation in IEEE 802.11n

The Viterbi decoder in IEEE 802.11n standard is implemented to efficiently decode the convolutionally encoded data received over the wireless channel. The standard specifies certain constraints and optimizations to ensure fast and accurate decoding, such as using soft-decision decoding through LLR.

The convolutional code used in IEEE 802.11n has a constraint length of 7, with generator polynomials typically defined as [133 171] in octal. This setup forms a

trellis structure with $2^6 = 64$ states, requiring 6 steps to transition from the initial state to potentially all other states. The Viterbi algorithm operates by calculating path metrics, which represent the cumulative metric of the most likely sequence of states leading to each state in the trellis.

Soft Decision Decoding

Soft-decision decoding, as implemented in IEEE 802.11n, utilizes LLRs to quantify the confidence in each bit's value. Unlike hard-decision decoding, which only considers the most likely bit value, soft-decision decoding provides more precise information by taking into account the probability of received bits being a '0' or '1'. This results in improved error correction performance, as the Viterbi algorithm can more accurately assess the likelihood of different paths through the trellis.

In the context of IEEE 802.11n, LLRs are the input to the Viterbi decoder. The Viterbi decoder processes these LLRs to determine the most likely transmitted bit sequence, minimizing the probability of error. The output of this process is the decoded bits, which form the final corrected data stream.

4.3 Leveraging Channel Information

In this section, we describe an enhanced Viterbi decoding algorithm that utilizes channel state information to improve decoding accuracy and overall system efficiency. By taking advantage of the orthogonal subcarriers used in OFDM, we consider the unique channel characteristics experienced by each subcarrier. In the IEEE 802.11n SISO system, there are 52 data subcarriers which our algorithm analyzes

individually to optimize performance. This method, implemented in the function `wlanBCCDecodeplus`, integrates channel state information into the convolutional decoding process, providing a more informed and adaptive approach compared to traditional methods.

4.3.1 Overview of the Algorithm

The algorithm consists of the following key steps:

Initialization

- `Trellis Construction`: Use a `poly2trellis` structure based on a convolutional code with constraint length 7 and polynomials [133 171]. This defines the state transitions and outputs for the Viterbi algorithm.
- `Path Metrics and Survivor Paths`: Initialize path metrics (cumulative metrics for each state) and survivor paths (paths leading to each state) for all states. The initial metric for all states is set to infinity except for the zero state, which starts with zero.

Channel Information Integration

- `Channel State Application`: The channel state information, represented by a vector h, is permuted according to the given index mapping determined by the interleaver. This permuted vector is then used to adjust the received symbols based on the channel's effect.

- `Subcarrier Processing`: For each received LLR, the corresponding channel coefficients are used to compute the Euclidean distances between the received symbol and all possible transmitted symbols. This distance calculation incorporates the channel information to better estimate the transmitted symbols.

Decoding with Channel Information

For each time step, the algorithm performs the following:

- `Compute the Branch Metrics`: For each possible state transition, the algorithm calculates the branch metrics, which represent the likelihood of a transition given the received symbols. Specifically, the branch metric is computed as the Euclidean distance between the received symbol and the expected symbol constellation point (adjusted for channel effects). This distance measures how close the received symbol is to each possible transmitted symbol, with smaller distances indicating a higher likelihood of the corresponding transmitted symbol. The channel coefficients, which vary across subcarriers, are used to weight the received symbols accordingly, accounting for channel effects such as fading and noise.

- `Update the Path Metrics`: Using the computed branch metrics, the algorithm updates the path metrics by considering the possible state transitions and their associated metrics. For each state, the path metric is updated by selecting the minimum metric path, which corresponds to the most likely sequence of state transitions leading to that state. This minimum metric path is known as the survivor path, and it is recorded for each state at each time step.

The process of selecting the path with the lowest cumulative metric ensures that the most probable sequence of transmitted symbols is identified.

Traceback

After processing all received symbols, the traceback step reconstructs the most likely sequence of states that led to the observed sequence. This is done by tracing back from the state with the minimum metric at the final time step, using the survivor paths recorded during the forward pass. The final decoded output is obtained by extracting the bits associated with the states along the traced path.

4.3.2 Advantages of Channel-State-Assisted Decoding

This approach enhances decoding performance by dynamically adjusting the decoding process based on real-time channel conditions. By incorporating channel state information, the algorithm can better account for distortions and noise introduced by the channel, leading to more accurate symbol recovery and overall improved system performance. Similar benefits have been explored in the context of joint source-protocol-channel decoding, as discussed in Dikici et al. (2011), where integrating multiple layers of information was shown to improve receiver performance in IEEE 802.11n systems. Akay and Ayanoglu (2004) demonstrate how simplified metrics for bit-interleaved coded modulation (BICM) in M-ary QAM systems can reduce decoder complexity without compromising performance.

This method is particularly beneficial in wireless communication environments where channel conditions can vary rapidly. In our implementation and testing, we

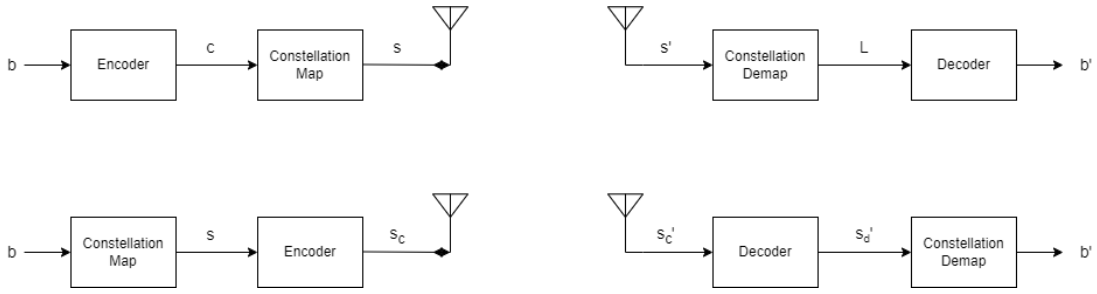

Figure 4.1: BPSK vs QAM decoder.

applied this channel-state-assisted decoding technique as an IEEE 802.11n decoder. The results demonstrated improvements in decoding accuracy and robustness, especially in scenarios with fluctuating channel conditions. The use of channel state information allowed for more precise adjustments during the decoding process, thereby enhancing the resilience of the system against channel impairments such as fading and interference.

4.3.3 BPSK vs QAM decoder

In the context of IEEE802.11n SISO systems, the decoding strategy can vary depending on the MCS. Fig. 4.1 illustrates two different decoder architectures: the upper system corresponds to BPSK decoding, while the lower system represents a QAM decoder.

Binary Decoder System The upper part of the diagram represents a BPSK decoder system, where b refers to the binary data before encoding. The encoded data is represented by c, which is the binary-coded version of b, enhancing its resilience to noise. The modulated signal after the constellation mapping is denoted as s, which

is then transmitted. At the receiver side, the LLR, denoted by $L \in \Re$, is computed based on the received signal *s'* and quantifies the likelihood of a bit being a 0 or 1. This LLR is provided to the binary decoder to obtain the decoded data *b'*.

QAM Decoder System The lower part of the diagram illustrates a QAM decoder system. In this setup, the process begins with mapping the binary data, b, onto the QAM constellation, resulting in the modulated signal s. The modulated signal s is then encoded, producing the encoded s_c, which is then transmitted over the communication channel. At the receiver side, the received signal s'_c is decoded into complex decoded symbols s'_d, which are then demapped to obtain the original binary data *b'*.

The QAM system, while capable of higher data rates, requires more complex algorithms for decoding due to its larger and more complex constellation. This increases the computational burden and the potential for errors, especially in noisy environments.

In contrast, the binary system, as shown in the upper part of the diagram, is simpler and more robust, making it a preferred choice in scenarios where simplicity and reliability are prioritized over data rate. The simplicity arises from its binary nature, which allows for straightforward constellation mapping and demapping, as well as LLR computation.

Thus, due to the higher complexity and the associated computational and implementation challenges of QAM, the binary decoding system is implemented in IEEE802.11n.

Mathematical Processing

5.1 Log Likelihood Ratio

For soft output constellation demapping, the LLR for each i-th bit is computed as

$$L(c_i) = \log \frac{\sum_{s \in \mathcal{S}_0^i} f_{\tilde{Y}|X=s}(\tilde{y}) p(X=s)}{\sum_{s \in \mathcal{S}_1^i} f_{\tilde{Y}|X=s}(\tilde{y}) p(X=s)}, \tag{5.1}$$

where $\mathcal{S}_1^i$ and $\mathcal{S}_0^i$ are the sets of constellation points for which the i-th bit is 1 and 0, respectively, Cao et al. (2019).

Assuming $p(X=s) = \frac{1}{M}$, where M is the modulation order, we have

$$L(c_i) = \log \frac{\sum_{s \in \mathcal{S}_0^i} f_{\tilde{Y}|X=s}(\tilde{y})}{\sum_{s \in \mathcal{S}_1^i} f_{\tilde{Y}|X=s}(\tilde{y})}. \tag{5.2}$$

Given that $f_{\tilde{Y}|X=s}(\tilde{y}) = \frac{1}{\sqrt{\pi \sigma_{\tilde{z}}^2}} \exp\left(-\frac{|\tilde{y}-s|^2}{\sigma_{\tilde{z}}^2}\right)$, the LLR can be simplified to

$$L(c_i) = \log \left(\frac{\sum_{s \in \mathcal{S}_0^i} \exp\left(-\frac{|\tilde{y}-s|^2}{\sigma_{\tilde{z}}^2}\right)}{\sum_{s \in \mathcal{S}_1^i} \exp\left(-\frac{|\tilde{y}-s|^2}{\sigma_{\tilde{z}}^2}\right)} \right). \tag{5.3}$$

This demapper, which calculates an LLR, is known as the LogAPP (logarithmic a-posteriori probability or LogMAP) demapper, Land et al. (2004). Due to the complicated mathematical operations of LogAPP, the max-log approximation LLR algorithm is used. The MaxLog demapper is an algorithm that calculates LLR by using only the two closest constellation points with the bit value at the given bit

position, Robertson et al. (1995). Using the log-sum exponential approximation:

$$\log\left(\sum_i e^{\phi_i}\right) \approx \max_i(\phi_i), \tag{5.4}$$

the LLR can be approximated as:

$$L(c_i) \approx \log\left(\frac{\exp\left(-\frac{\min_{s\in\mathcal{S}_0^i}|\tilde{y}-s|^2}{\sigma_{\tilde{z}}^2}\right)}{\exp\left(-\frac{\min_{s\in\mathcal{S}_1^i}|\tilde{y}-s|^2}{\sigma_{\tilde{z}}^2}\right)}\right) = \frac{1}{\sigma_{\tilde{z}}^2}\left(\min_{s\in\mathcal{S}_1^i}|\tilde{y}-s|^2 - \min_{s\in\mathcal{S}_0^i}|\tilde{y}-s|^2\right). \tag{5.5}$$

The result is a value indicating whether that bit is more likely to be one or zero. For each input symbol to constellation demapper, the output is N_{bpscs} real numbers.

5.1.1 BPSK

In the frequency domain, after the OFDM demodulation, let s be the transmitted signal and h the channel impulse response. For BPSK, as shown in Fig. 5.1, the transmitted symbols s are binary values mapped to $\{+1, -1\}$ corresponding to $\{1, 0\}$. The received signal y can be expressed as

$$y = hs + z, \tag{5.6}$$

where z represents the additive white gaussian noise (AWGN) with mean 0 and variance σ_z^2.

After the equalization, which aims to compensate for the channel effects, the

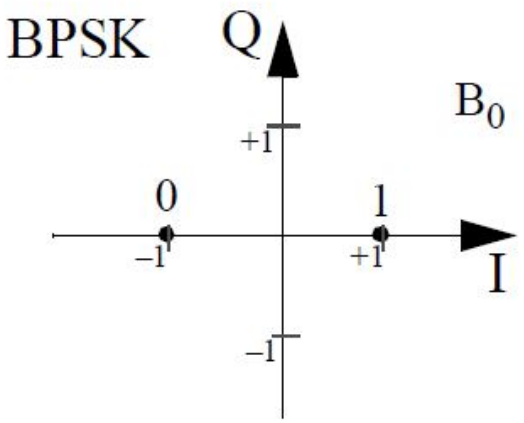

Figure 5.1: BPSK Constellation Mapping.

received signal becomes

$$\tilde{y} = \frac{y}{h} = s + \frac{z}{h} = s + \tilde{z}. \tag{5.7}$$

The variance of the noise term $\tilde{z}$ after the equalization is

$$\sigma_{\tilde{z}}^2 = \frac{\sigma_z^2}{|h|^2}. \tag{5.8}$$

In the case of BPSK modulation, we know that $\mathcal{S}_0 = \{-1\}$, $\mathcal{S}_1 = \{+1\}$. Thus,

substituting into the expression of LLR (5.5) we get

$$\begin{aligned} L(c_i) &= \frac{1}{\sigma_{\tilde{z}}^2}\left(|\tilde{y}-1|^2 - |\tilde{y}+1|^2\right) \\ &= \frac{1}{\sigma_{\tilde{z}}^2}\left(\left(\Re\{\tilde{y}\}-1\right)^2 + \left(\Im\{\tilde{y}\}\right)^2 - \left(\Re\{\tilde{y}\}+1\right)^2 - \left(\Im\{\tilde{y}\}\right)^2\right) \\ &= \frac{1}{\sigma_{\tilde{z}}^2}\left(\left(\Re\{\tilde{y}\}-1\right)^2 - \left(\Re\{\tilde{y}\}+1\right)^2\right) \\ &= -\frac{4\Re\{\tilde{y}\}}{\sigma_{\tilde{z}}^2}. \end{aligned} \tag{5.9}$$

Now, using $\tilde{y} = s + \tilde{z}$, we can write

$$\begin{aligned} L(c_i) &= -\frac{4\Re\{s+\tilde{z}\}}{\sigma_{\tilde{z}}^2} \\ &= -\frac{4\Re\{s\}}{\sigma_{\tilde{z}}^2} - \frac{4\Re\{\tilde{z}\}}{\sigma_{\tilde{z}}^2} \\ &= -\frac{4s}{\sigma_{\tilde{z}}^2} - \frac{4\Re\{\tilde{z}\}}{\sigma_{\tilde{z}}^2}. \end{aligned} \tag{5.10}$$

Since $\tilde{z} = \frac{z}{h}$, we can rewrite the expression as

$$L(c_i) = -4\frac{|h|^2}{\sigma_z^2}s - 4w, \tag{5.11}$$

where $w = \frac{\Re\{\tilde{z}\}}{\sigma_{\tilde{z}}^2}$ with variance

$$\sigma_w^2 = \frac{1}{2\sigma_{\tilde{z}}^2}. \tag{5.12}$$

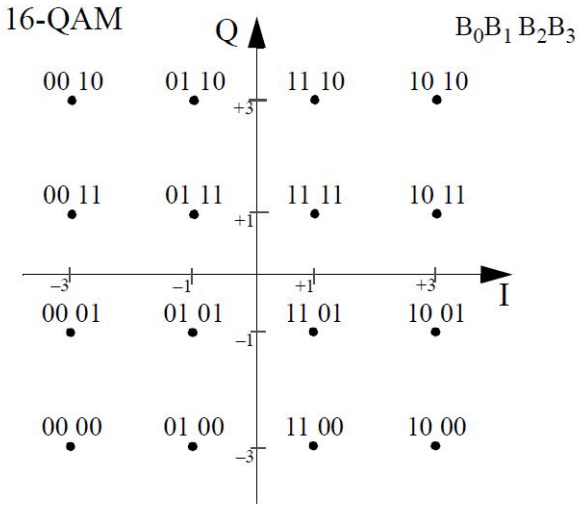

Figure 5.2: QAM Constellation Mapping

5.1.2 16-QAM

For 16-QAM, the transmitted symbols x are complex values representing one of the 16 constellation points, as shown in Fig. 5.2. The received signal y after the equalization can be expressed, as in the case of BPSK, by (5.7).

If we suppose s_0^i to be the symbol that minimizes the distance $\min_{s\in\mathcal{S}_0^i} |\tilde{y}-s|^2$ and s_1^i to be the symbol that minimizes the distance $\min_{s\in\mathcal{S}_1^i} |\tilde{y}-s|^2$, the expression for the LLR (5.5) is simplified to

$$L(c_i) = \frac{1}{\sigma_{\tilde{z}}^2}(|\tilde{y}-s_1^i|^2 - |\tilde{y}-s_0^i|^2). \tag{5.13}$$

Because the expression involves complex numbers, we can write that

$$|\tilde{y}-s|^2 = (\tilde{y}-s)(\tilde{y}-s)^*, \tag{5.14}$$

where $(\tilde{y}-s)^*$ denotes the complex conjugate transpose of $(\tilde{y}-s)$. Since $(\tilde{y}-s)^* = \tilde{y}^* - s^*$, we have

$$\begin{aligned} |\tilde{y}-s|^2 &= (\tilde{y}-s)(\tilde{y}^*-s^*) \\ &= (\tilde{y}\tilde{y}^* - \tilde{y}s^* - s\tilde{y}^* + ss^*) \\ &= (|\tilde{y}|^2 - \tilde{y}s^* - s\tilde{y}^* + |s|^2) \\ &= (|\tilde{y}|^2 - 2\Re\{\tilde{y}s^*\} + |s|^2). \end{aligned} \tag{5.15}$$

So, the expression for LLR can be written as

$$\begin{aligned} L(c_i) &= \frac{1}{\sigma_{\tilde{z}}^2}(|s_1^i|^2 - |s_0^i|^2 + 2\Re\{\tilde{y}s_0^{i\,*}\} - 2\Re\{\tilde{y}s_1^{i\,*}\}) \\ &= \frac{1}{\sigma_{\tilde{z}}^2}\left(|s_1^i|^2 - |s_0^i|^2 + 2\Re\left\{\tilde{y}(s_0^{i\,*} - s_1^{i\,*})\right\}\right). \end{aligned} \tag{5.16}$$

We denote the term $\Delta s_i = s_0^i - s_1^i$ with phase $\varphi_i = \angle \Delta s_i$ and the expression becomes

$$\begin{aligned} L(c_i) &= \frac{1}{\sigma_{\tilde{z}}^2}\left(|s_1^i|^2 - |s_0^i|^2 + 2\Re\left\{\tilde{y}\Delta s_i^*\right\}\right) \\ &= \frac{1}{\sigma_{\tilde{z}}^2}\left(|s_1^i|^2 - |s_0^i|^2 + 2\Re\left\{\tilde{y}\,|\Delta s_i|\,e^{-j\varphi_i}\right\}\right) \\ &= \frac{|s_1^i|^2 - |s_0^i|^2}{\sigma_{\tilde{z}}^2} + \frac{2\,|\Delta s_i|}{\sigma_{\tilde{z}}^2}\Re\left\{\tilde{y}e^{-j\varphi_i}\right\}. \end{aligned} \tag{5.17}$$

Since $\tilde{y} = s + \tilde{z}$ the above expression for LLR can be rewritten as

$$\begin{aligned} L(c_i) &= \frac{|s_1^i|^2 - |s_0^i|^2}{\sigma_{\tilde{z}}^2} + \frac{2\,|\Delta s_i|}{\sigma_{\tilde{z}}^2}\Re\left\{se^{-j\varphi_i}\right\} + \frac{2\,|\Delta s_i|}{\sigma_{\tilde{z}}^2}\Re\left\{\tilde{z}e^{-j\varphi_i}\right\} \\ &= \frac{|s_1^i|^2 - |s_0^i|^2}{\sigma_{\tilde{z}}^2} + \frac{2\,|\Delta s_i|}{\sigma_{\tilde{z}}^2}\Re\left\{se^{-j\varphi_i}\right\} + \frac{|\Delta s_i|}{\sigma_{\tilde{z}}}\Re\left\{2\frac{\tilde{z}}{\sigma_{\tilde{z}}}e^{-j\varphi_i}\right\}. \end{aligned} \tag{5.18}$$

5.2 Log Likelihood Ratio with Constant Noise Variance

5.2.1 BPSK

To standardize the noise variance, we multiply (5.11) by the standard deviation of $\tilde{z}$

$$\tilde{L}(c_i) = \sigma_{\tilde{z}_i} L(c_i) = -4\frac{|h_i|}{\sigma_{z_i}} s_i - 4\tilde{w}_i, \tag{5.19}$$

where $\tilde{w}_i = \frac{\Re\{\tilde{z}_i\}}{\sigma_{\tilde{z}}}$ and its variance $\sigma^2_{\tilde{w}_i} = \frac{1}{2}$.

The input to our decoder is $\tilde{L}(c_i)$, and the Viterbi decoding algorithm incorporates the coefficient $-4\frac{|h_i|}{\sigma_{z_i}}$, when computing the Euclidean distances.

5.2.2 16-QAM

Each symbol s is associated with four LLRs, denoted as $L(c_i)$, for $i \in \{1, 2, 3, 4\}$.

$$\tilde{L}(c_i) = \frac{|s_1^i|^2 - |s_0^i|^2}{|\Delta s_i|\,\sigma_{\tilde{z}}} + \frac{1}{\sigma_{\tilde{z}}} \Re\left\{2se^{-j\varphi_i}\right\} + \Re\left\{2\frac{\tilde{z}}{\sigma_{\tilde{z}}} e^{-j\varphi_i}\right\}. \tag{5.20}$$

We know that if the decoded bit $c_i = 1$, then the corresponding mapped bit $b_i = 1$ and the signal symbol $s = s_1^i$, where s_1^i is the constellation point closest to the received signal when the i-th bit is 1. Conversely, if the decoded bit $c_i = 0$, then the mapped bit $b_i = -1$ and the signal symbol $s = s_0^i$, where s_0^i is the constellation point closest to the received signal when the i-th bit is 0.

The above observation results in the below expression for s

$$s = \frac{s_1^i + s_0^i}{2} + b_i \frac{s_1^i - s_0^i}{2}. \tag{5.21}$$

Then, the real part of $2s$ can be expressed as

$$\Re\{2s\} = \Re\left\{s_1^i + s_0^i\right\} + b_i \Re\left\{s_1^i - s_0^i\right\}. \tag{5.22}$$

For the first bit, $i = 1$, based on observation and the constellation mapping, we find that s_0^1 and s_1^1 are in the same row, having identical imaginary parts. Furthermore, $\Re\{s_0^1\} < \Re\{s_1^1\}$. It is important to note that $\Delta s_1 = \Re\{s_0^1\} - \Re\{s_1^1\} < 0$, which indicates that $\varphi_1 = \pi$. From (5.20), we obtain

$$\begin{aligned}\tilde{L}(c_1) &= \frac{\Re^2\{s_1^1\} - \Re^2\{s_0^1\}}{|\Re\{s_0^1\} - \Re\{s_1^1\}|\,\sigma_{\tilde{z}}} + \frac{1}{\sigma_{\tilde{z}}}\Re\left\{2se^{-j\pi}\right\} + \Re\left\{2\frac{\tilde{z}}{\sigma_{\tilde{z}}}e^{-j\pi}\right\} \\ &= \frac{\Re\{s_1^1 + s_0^1\}}{\sigma_{\tilde{z}}} - \frac{1}{\sigma_{\tilde{z}}}\Re\{2s\} - 2\Re\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\}.\end{aligned} \tag{5.23}$$

Substituting (5.22) we get

$$\tilde{L}(c_1) = -\frac{\Re\{s_1^1 - s_0^1\}}{\sigma_{\tilde{z}}} b_1 - 2\Re\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\}. \tag{5.24}$$

For the second bit, $i = 2$, the constellation shows that $\Re\{s_0^2\} < \Re\{s_1^2\}$ when $\Re\{\tilde{y}\} < 0$, while $\Re\{s_0^2\} > \Re\{s_1^2\}$ when $\Re\{\tilde{y}\} > 0$. Additionally, s_0^2 and s_1^2 are in the same row, indicating they share the same imaginary part. It is important to note that $\varphi_2 = \pi$ when $\Re\{\tilde{y}\} < 0$ and $\varphi_2 = 0$ when $\Re\{\tilde{y}\} > 0$. From (5.20), we obtain

$$\begin{aligned}\tilde{L}(c_2) &= \frac{\Re^2\{s_1^2\} - \Re^2\{s_0^2\}}{|\Re\{s_0^2\} - \Re\{s_1^2\}|\,\sigma_{\tilde{z}}} + \frac{1}{\sigma_{\tilde{z}}}\Re\left\{2se^{-j\varphi_2}\right\} + \Re\left\{2\frac{\tilde{z}}{\sigma_{\tilde{z}}}e^{-j\varphi_2}\right\} \\ &= \frac{\Re\{s_1^2 + s_0^2\}}{-\operatorname{sgn}(\Im\{\tilde{y}\})\,\sigma_{\tilde{z}}} + \frac{2}{\sigma_{\tilde{z}}}\Re\{s\operatorname{sgn}(\Re\{\tilde{y}\})\} + 2\Re\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\operatorname{sgn}(\Re\{\tilde{y}\})\right\} \\ &= \operatorname{sgn}(\Re\{\tilde{y}\})\left(-\frac{\Re\{s_1^2 + s_0^2\}}{\sigma_{\tilde{z}}} + \frac{1}{\sigma_{\tilde{z}}}\Re\{2s\} + 2\Re\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\}\right).\end{aligned} \tag{5.25}$$

Substituting (5.22) we get

$$\tilde{L}(c_2) = -\operatorname{sgn}(\Re\{\tilde{y}\})\frac{\Re\{s_1^2 - s_0^2\}}{\sigma_{\tilde{z}}}b_2 + \operatorname{sgn}(\Re\{\tilde{y}\})\,2\Re\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\}. \tag{5.26}$$

For the third bit, $i = 3$, the constellation indicates that s_0^3 and s_1^3 belong to the same column, thus having identical real parts. We observe that $\Im\{s_0^3\} < \Im\{s_1^3\}$, leading to $\Delta s_3 = \Im\{s_0^3\} - \Im\{s_1^3\} < 0$, which implies $\varphi_3 = \frac{\pi}{2}$. From (5.20), we obtain

$$\begin{aligned}\tilde{L}(c_3) &= \frac{\Im^2\{s_1^3\} - \Im^2\{s_0^3\}}{|\Im\{s_0^3\} - \Im\{s_1^3\}|\,\sigma_{\tilde{z}}} + \frac{1}{\sigma_{\tilde{z}}}\Re\left\{2se^{-j\frac{\pi}{2}}\right\} + \Re\left\{2\frac{\tilde{z}}{\sigma_{\tilde{z}}}e^{-j\frac{\pi}{2}}\right\} \\ &= \frac{\Im\{s_1^3 + s_0^3\}}{\sigma_{\tilde{z}}} - \frac{2}{\sigma_{\tilde{z}}}\Re\{-js\} - 2\Re\left\{-j\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\} \\ &= \frac{\Im\{s_1^3 + s_0^3\}}{\sigma_{\tilde{z}}} - \frac{2}{\sigma_{\tilde{z}}}\Im\{s\} - 2\Im\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\}.\end{aligned} \tag{5.27}$$

Substituting (5.22) we get

$$\tilde{L}(c_3) = -\frac{\Im\{s_1^3 - s_0^3\}}{\sigma_{\tilde{z}}}b_3 - 2\Im\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\}. \tag{5.28}$$

For the fourth bit, $i = 4$, the constellation indicates that s_0^3 and s_1^3 are in the

same column, thus sharing the same real part. We observe that $\Im\{s_0^4\} < \Im\{s_1^4\}$ when $\Im\{\tilde{y}\} < 0$, which corresponds to $\varphi_4 = \frac{\pi}{2}$. Conversely, $\Im\{s_0^4\} > \Im\{s_1^4\}$ when $\Im\{\tilde{y}\} > 0$, indicating $\varphi_4 = -\frac{\pi}{2}$. From (5.20), we obtain

$$\begin{aligned}\tilde{L}(c_4) &= \frac{\Im^2\{s_1^4\} - \Im^2\{s_0^4\}}{|\Im\{s_0^4\} - \Re\{s_1^4\}|\,\sigma_{\tilde{z}}} + \frac{1}{\sigma_{\tilde{z}}}\Re\left\{2se^{-j\varphi_4}\right\} + \Re\left\{2\frac{\tilde{z}}{\sigma_{\tilde{z}}}e^{-j\varphi_4}\right\} \\ &= \frac{\Im\{s_1^4 + s_0^4\}}{\operatorname{sgn}(\Im\{\tilde{y}\})\,\sigma_{\tilde{z}}} + \frac{2}{\sigma_{\tilde{z}}}\Re\{sj\operatorname{sgn}(\Im\{\tilde{y}\})\} + 2\Re\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}j\operatorname{sgn}(\Im\{\tilde{y}\})\right\} \\ &= \operatorname{sgn}(\Im\{\tilde{y}\})\left(-\frac{\Im\{s_1^4 + s_0^4\}}{\sigma_{\tilde{z}}} + \frac{2}{\sigma_{\tilde{z}}}\Im\{s\} + 2\Im\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\}\right).\end{aligned} \tag{5.29}$$

Substituting (5.22) we get

$$\tilde{L}(c_4) = -\operatorname{sgn}(\Im\{\tilde{y}\})\frac{\Im\{s_1^4 - s_0^4\}}{\sigma_{\tilde{z}}}b_4 + \operatorname{sgn}(\Im\{\tilde{y}\})\,2\Im\left\{\frac{\tilde{z}}{\sigma_{\tilde{z}}}\right\} \tag{5.30}$$

The computed values $\tilde{L}(c_i)$ serve as the input to our decoder, where each $\tilde{L}(c_i)$ represents the LLR for the i-th bit. The coefficient corresponding to b_i is the amplitude provided to the decoder. This amplitude influences the decoder's decision on the value of b_i, ensuring accurate decoding of the transmitted data.

We observe that 5.24-5.26 and 5.28-5.30 have the same noise variance, which means the corresponding LLRs are correlated. This leads to the assumption that we could achieve more efficient decoding by implementing a more complex trellis for the decoding process and decoding adjacent bits together. Although this is not part of this thesis, where we focus on studying the efficiency of incorporating channel information in the decoding process, it is worth noting. The mathematical frame-

work and processing of LLRs detailed in this chapter provide a valuable foundation for future work. Our results suggest that while the presented approach does not outperform existing methodologies, it highlights potential areas for improvement in decoding algorithms. This aligns with similar methodologies found in the literature, such as Tosato and Bisaglia (2002) and Mao et al. (2016).

Comparison

In this chapter, we compare the performance of the standard Viterbi decoder with our proposed channel-state-assisted convolutional decoder in an IEEE 802.11n SISO system. The SISO configuration involves a single antenna for transmission and reception, making it a fundamental setup in wireless communication systems. The transmitter uses BCC encoder with a coding rate of 1/2 to protect the transmitted data. The MCS considered are BPSK and 16-QAM.

The performance is evaluated based on the bit error rate (BER) versus signal-to-noise ratio (SNR) for the different modulation schemes under varying channel conditions. The primary metric for comparison is the BER, which indicates the error rate in received bits. The tests were conducted under varying SNR levels and the results of our experiments are presented in the following figures. Each figure shows the BER performance for a given scenario.

6.1 BPSK

Fig. 6.1 shows the BER vs. SNR performance for BPSK modulation in a standard channel scenario. As the SNR increases, the BER decreases for both decoders, as expected. However, the BER curve for the channel-state-assisted decoder starts to diverge from that of the standard Viterbi decoder, particularly at lower SNRs. This indicates a better performance of the channel-state-assisted decoder, which consistently achieves a lower BER. This improvement is attributed to the use of channel state information derived from LLR $\tilde{L}$ of (5.19), which allows the decoder to adjust for channel distortions more effectively, enhancing the reliability of the

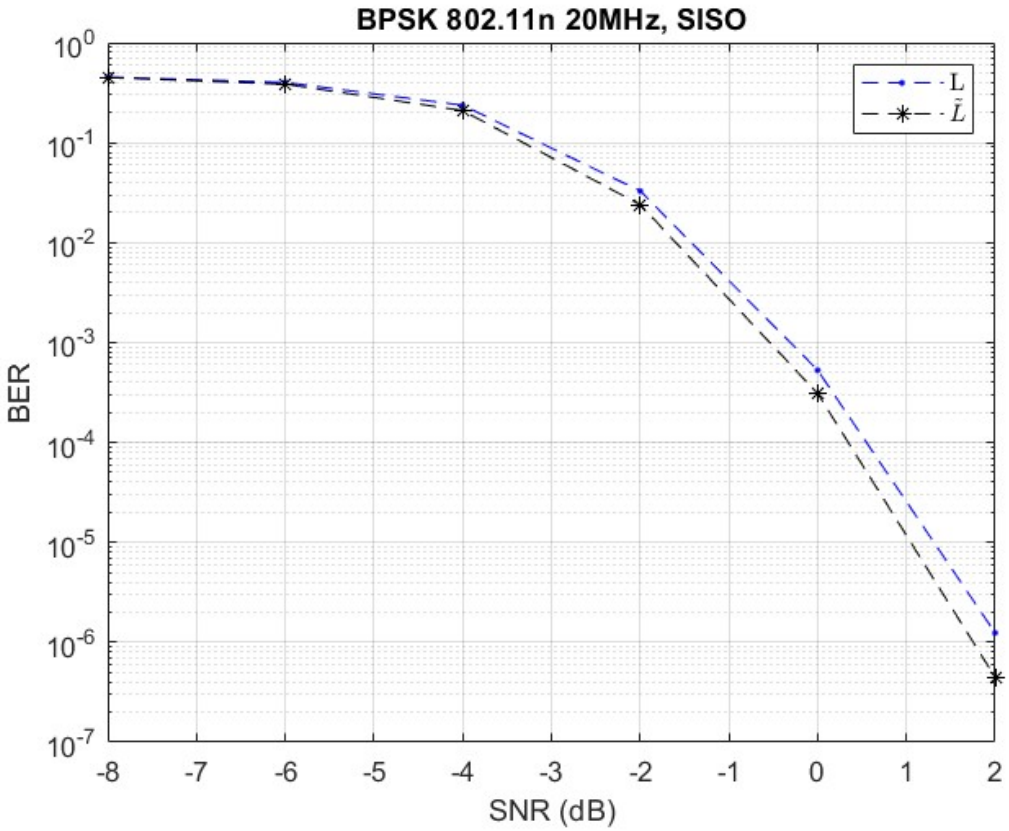

Figure 6.1: BER vs. SNR for BPSK Modulation.

decoding process.

6.2 BPSK under Modified Channel Conditions

Figs. 6.2 and 6.3 present the BER vs. SNR performance under modified channel conditions. These figures demonstrate an even greater performance disparity between the standard Viterbi decoder and the channel-state-assisted decoder.

The divergence in the BER curves is more pronounced in these modified channel scenarios, reflecting the robustness of the channel-state-assisted decoder to more

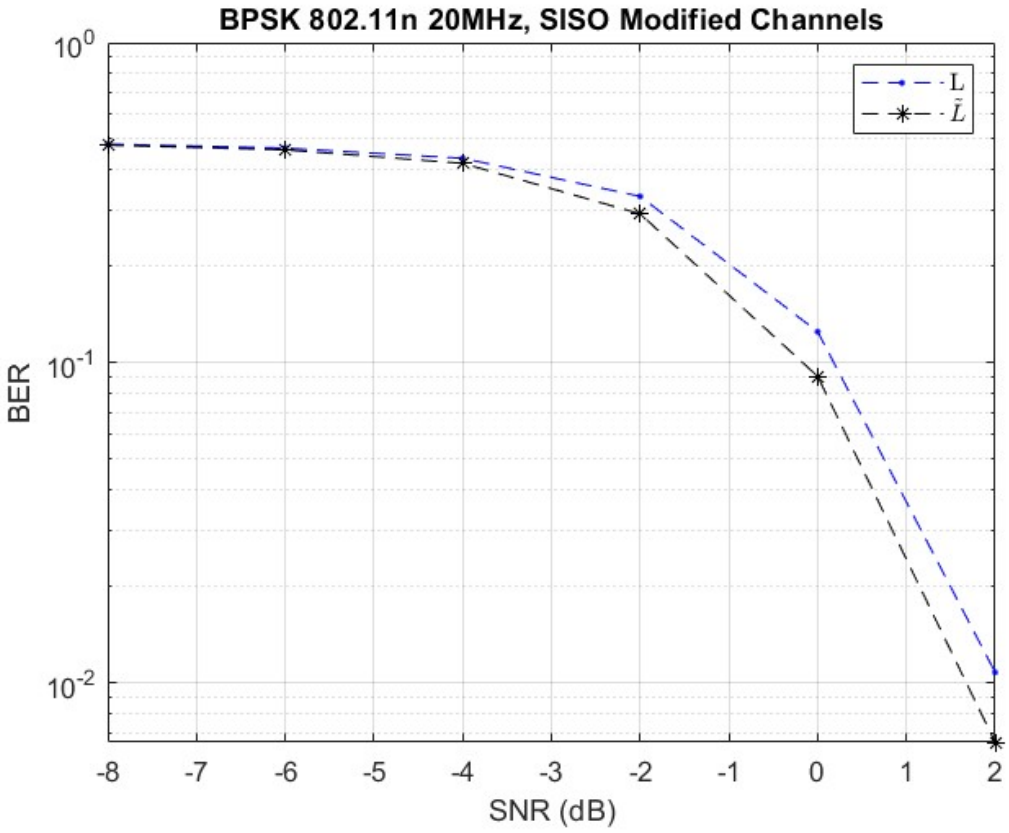

Figure 6.2: BER vs. SNR for BPSK Modulation Mild Modification.

severe channel impairments. This is a direct result of incorporating the $\tilde{L}$ LLR, as input in our decoder, which provides a more accurate representation of the likelihood ratios under varying channel conditions due to the constant noise variance term. By utilizing the channel coefficient—reflecting the specific channel characteristics due to OFDM—the decoder can make more precise bit decisions. Consequently, this improved accuracy decreases the BER.

These results highlight the advantages of the channel-state-assisted approach in real-world scenarios where channel conditions can vary unpredictably. The ability

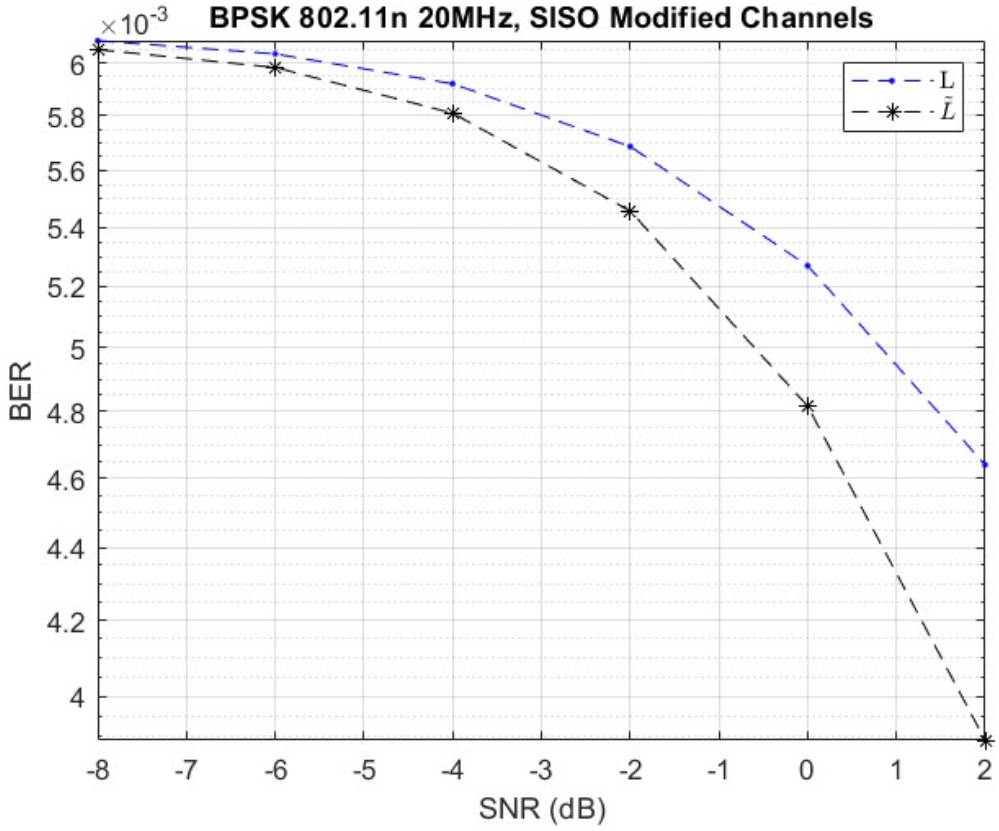

Figure 6.3: BER vs. SNR for BPSK Modulation Extreme Modification.

to adapt to such conditions is crucial for maintaining high-quality communication links, especially in dynamic wireless environments.

6.3 16-QAM

Fig. 6.4 shows the BER vs. SNR performance for 16-QAM modulation in a standard channel scenario. The BER performance for our proposed decoder and the built-in Viterbi decoder is identical, indicating that both perform equivalently.

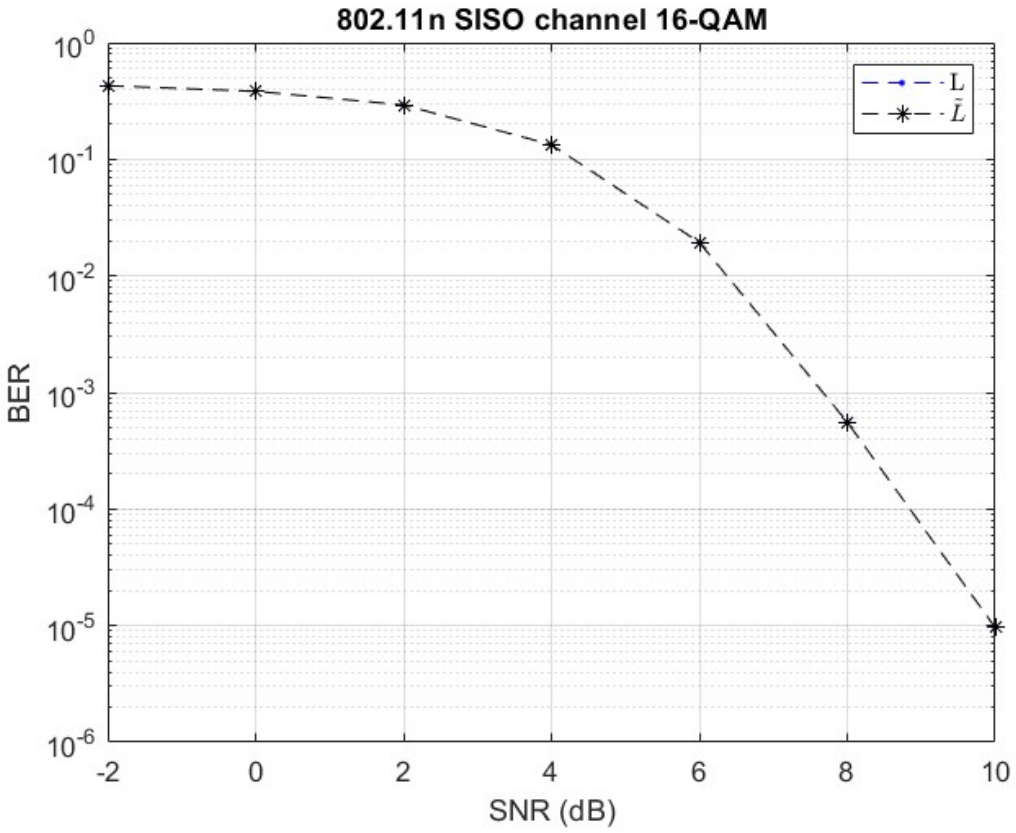

Figure 6.4: BER vs. SNR for 16-QAM Modulation.

6.4 Observations

In the case of BPSK modulation, our proposed decoder performs better than the standard Viterbi decoder. However, for 16-QAM modulation, the performance is equivalent. This equivalence can be attributed to the common noise term that affects the decoder, as described by 5.24-5.26 and 5.28-5.30. The common noise term likely impacts the performance, because the LLRs are correlated and should be decoded together for better performance.

Conclusion

In this thesis, we implemented and simulated SISO IEEE 802.11n links, focusing on the examination of channel-state-assisted convolutional decoding to improve system performance. Through detailed derivations, we developed closed-form approximations for the LLR of the coded bits at the receiver end, considering the specific signal constellation of the IEEE 802.11n standard. Our simulations were performed over IEEE TGn Model-B channels, and we compared the performance of our proposed channel-state-assisted decoder with the standard Viterbi decoder.

For BPSK, the results demonstrated that our proposed algorithm performs better than the built-in Viterbi decoder. This improvement can be attributed to the use of channel state information that allows the decoder to adjust more accurately for channel conditions, resulting in a lower BER.

In contrast, for 16-QAM, our proposed algorithm achieved the same BER performance as the built-in Viterbi decoder. This equivalence is likely due to the common noise term variance associated with each bit in the 16-QAM constellation. Adjacent bits carry common noise variance, and considering it jointly in the decoding process might be necessary to achieve better performance.

In conclusion, while our channel-state-assisted decoding algorithm significantly improves performance for BPSK modulation, the same enhancement is not observed for 16-QAM due to the common noise term variance. Future work could focus on developing strategies to implement a more complex trellis structure that takes this common noise variance into consideration, to decode the received bits more efficiently through LLRs and channel coefficients.

Bibliography

E. Akay and E. Ayanoglu. Low complexity decoding of bit-interleaved coded modulation for m-ary qam. In *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, volume 2, pages 901–905 Vol.2, 2004. doi: 10.1109/ICC.2004.1312632.

Xianle Cao, Yi Liu, and Dongfang Hu. Simplified llr algorithm for m-qam demodulation. *The Journal of Engineering*, 2019(21):7370–7375, 2019. doi: https://doi.org/10.1049/joe.2019.0634. URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/joe.2019.0634.

Pramod Viswanath David Tse. *Fundamentals of Wireless Communication*. Cambridge University Press, 2005. ISBN 9780511807213.

Çağatay Dikici, Anissa Mokraoui, Michel Kieffer, and Pierre Duhamel. Joint source-protocol-channel decoding: Improving 802.11n receivers. In *2011 19th European Signal Processing Conference*, pages 1519–1523, 2011.

Matthew S Gast. *802.11 Wireless Networks: The Definitive Guide, Second Edition*. O'Reilly Media, Inc., 2005. ISBN 0596100523.

IEEE802.11n. Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications - corrigendum 1 – correct ieee 802.11ay assignment of protected announce support bit. *IEEE Std 802.11-2020/Cor 1-2022 (Corrigendum to IEEE Std 802.11-2020 as amended by IEEE Std 802.11ax-2021, IEEE Std 802.11ay-2021, and IEEE Std 802.11ba-2021)*, pages 1–18, 2022. doi: 10.1109/IEEESTD.2022.9999411.

G.Madhu kumar and A.Swetha A.Swetha. Design and implementation of convolution encoder and viterbi decoder. *International Journal of Scientific Research*, 1:65–66, 06 2012. doi: 10.15373/22778179/NOV2012/23.

Ingmar Land, Peter Hoeher, and Snjezana Gligorevic. Computation of symbol-wise mutual information in transmission systems with logapp decoders and application to exit charts. *ITG Fachbericht*, 195–202, 03 2004.

Emmanouil Logothetis. Efficient wlan link simulations by means of eesm, 10 2021.

Juquan Mao, Mahmoud Alfa Abdullahi, Pei Xiao, and Aijun Cao. A low complexity 256qam soft demapper for 5g mobile system. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 16–21, 2016. doi: 10.1109/EuCNC.2016.7560996.

J.G. Proakis and M. Salehi. *Contemporary Communication Systems Using MATLAB*. Electrical Engineering Series. PWS Pub., 1998. ISBN 9780534938048. URL https://books.google.gr/books?id=fhkfAQAAIAAJ.

P. Robertson, E. Villebrun, and P. Hoeher. A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain. In *Proceedings IEEE International Conference on Communications ICC '95*, volume 2, pages 1009–1013 vol.2, 1995. doi: 10.1109/ICC.1995.524253.

P. Kyritsi Vinko Erceg V. L. Schumacher. Ieee p802.11 wireless lans tgn channel models, 2004.

F. Tosato and P. Bisaglia. Simplified soft-output demapper for binary interleaved cofdm with application to hiperlan/2. In *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333)*, volume 2, pages 664–668 vol.2, 2002. doi: 10.1109/ICC.2002.996940.

William Tranter, K. Shanmugan, Theodore Rappaport, and Kurt Kosbar. *Principles of communication systems simulation with wireless applications*. Prentice Hall Press, USA, first edition, 2003. ISBN 0134947908.

A. van Zelst and T.C.W. Schenk. Implementation of a mimo ofdm-based wireless lan system. *IEEE Transactions on Signal Processing*, 52(2):483–494, 2004. doi: 10.1109/TSP.2003.820989.

A.J. Viterbi. An intuitive justification and a simplified implementation of the map decoder for convolutional codes. *IEEE Journal on Selected Areas in Communications*, 16(2):260–264, 1998. doi: 10.1109/49.661114.