DEMYSTIFYING WAVEFORM DESIGN AND SIGNAL PROCESSING FOR WIRELESS COMMUNICATION

Ng Kai Jiun Ian[1], Tan Beng Soon[2]
[1]Hwa Chong Institution, 661 Bukit Timah Road, Singapore 269734
[2]DSO National Laboratories, 12 Science Park Dr, Singapore 118225

Abstract
Digital communication lays the foundations for modern civilization and advancement and hence understanding of the topic is paramount. The multi-step process involving encoding and decoding, modulation and demodulation across an AWGN channel is simulated in order to analyse the performance of various schemes of modulation like BPSK, QPSK and 16-QAM. The performance with encoding methods like gray coding and (7,4) hamming code was tested. 16 QAM had the highest BER and both BPSK and QPSK had lower but identical BERs. BPSK with (7,4) Hamming Code had lower BER but only above a certain energy per bit threshold.

1. Introduction

Digital communication is the bedrock of modern civilization, facilitating the exchange and proliferation of information and knowledge. It has allowed information to transcend boundaries, allowing it to be accessed and sent from every corner of this globe. Hence, it has to come to serve as the foundation of the fourth industrial revolution. The pivotal role that digital communication plays in the continued advancement of humanity cannot be understated.

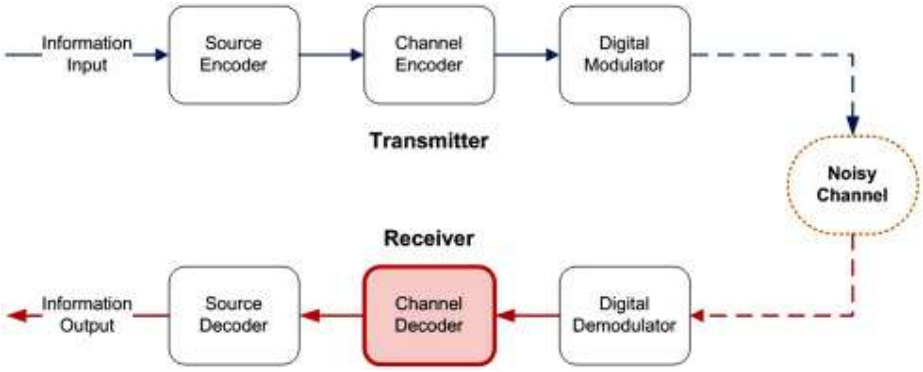

Fig. 1. Diagram of a digital communication system [1]

As illustrated in the diagram above, digital communication involves the transfer of data through the transmission of a signal from the transmitter to the receiver. Prior to transmission, the signal undergoes encoding and modulation by a modulator. Thereafter, the data is transmitted through a channel. Finally, the signals are decoded before being demodulated at the demodulator.

For the simulation that will be conducted, gray coding and error correcting code, specifically hamming code, will be implemented. In addition, the modulation schemes of Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK) and 16 - Quadrature

Amplitude Modulation (QAM) will be utilised. In addition, an Additive White Gaussian Noise (AWGN) channel will be used.

2. Materials and methods

2.1. Modulation and demodulation

Underlying digital communication is modulation, which is defined as the process of encoding a digital information signal into the amplitude, phase or frequency of the transmitted signal [2]. In layman terms, it is the process of packaging information in such a way that it can be transmitted. It involves the conversion of a bit or a string of bits into symbols on the in phase and quadrature axis on a constellation diagram. The symbols used to represent these bits are numbers for the in phase axis and imaginary numbers for the quadrature axis. For example, for QPSK, the bits 00 can be represented by 1 + 1j on the constellation diagram.

2.1.1. Binary Phase Shift Keying (BPSK)

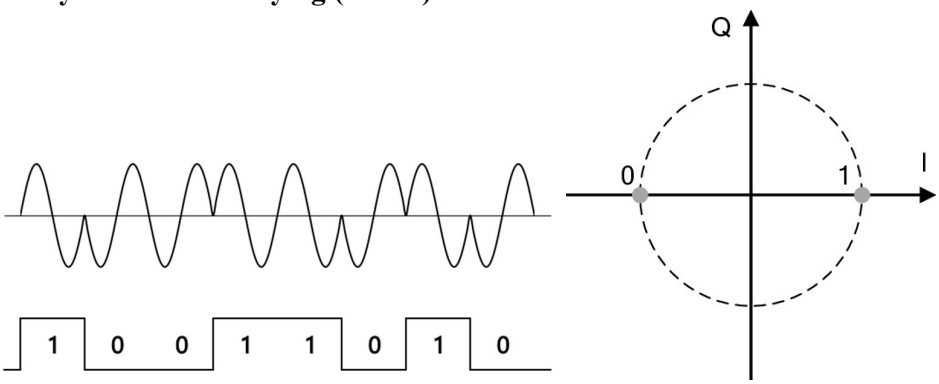

Fig. 2 & 3. Modulated Signals and their corresponding binary digits in BPSK [3] and the constellation diagram for BPSK [4]

BPSK is a dual phase modulation scheme whereby the signal is modulated by changing its phase by 180° for each binary symbol [5]. The binary digits of 0 and 1 are represented by phase shifts of 0° and 180° respectively. This phenomenon is illustrated in figures 2 and 3 above. The binary digits 0 and 1 are converted to the symbols -1 and 1 respectively. They are then demodulated into the digits 0 or 1 depending on whether the symbol is below or above 0 respectively after passing through the AWGN channel. The code for BPSK is as follows:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erfc

# Parameters
Eb_N0_dB = np.arange(-10, 12, 1)
Eb_N0 = 10**(Eb_N0_dB / 10)
num_pack = 100
pack_size = 100000
```

```
bits = np.random.randint(0, 2, (num_pack,pack_size))
symbols = [1, -1]
BER = np.zeros(len(Eb_N0_dB))

for i, eb_n0 in enumerate(Eb_N0):
    BER_sum = 0
    for pack in bits:
      # Modulation
      mod = np.array([symbols[bit] for bit in pack])

      # AWGN
      N0 = 1/(eb_n0)
      noise = np.sqrt(N0/2)*(np.random.randn(pack_size))
     # Receive symbols
      received = mod + noise

     # Demodulation
      demod = [0 if symbol > 0 else 1 for symbol in received]
      BER_pack = np.sum(np.abs(pack - demod))/len(pack)
      BER_sum += BER_pack

    # Calculate BER
    BER[i] = BER_sum / num_pack
```

2.1.2. Quadrature Phase Shift Keying (QPSK)

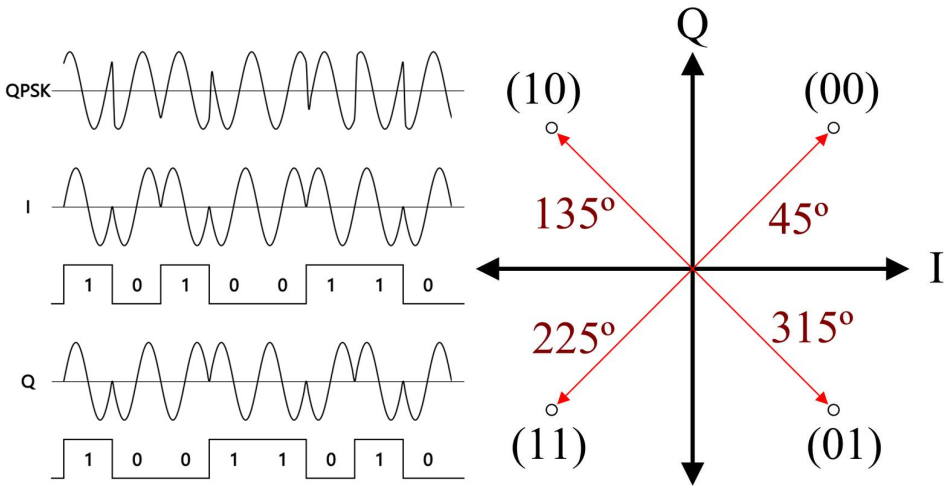

Fig. 4 & 5. Modulated Signals in the and their corresponding binary digits in QPSK [6] and the constellation diagram for QPSK [7]

QPSK, unlike BPSK, modulates 2 bits at once. The signal is modulated by changing its phase by 90° for each binary symbol. The binary digits of 00, 01, 10 and 11 are represented by the

phase shifts of 45°,135°,225° and 315° respectively [8]. This phenomenon is illustrated in figures 4 & 5 above. For every 2 bits, the first bit is converted to a symbol on the inphase axis (either -1 or 1) whereas the second bit is converted to a symbol on the quadrature axis (either -1j or 1j). The code for QPSK is as follows:

```
import numpy as np
import matplotlib.pyplot as plt

Eb_N0_dB = np.arange(-10, 14, 1)
Eb_N0 = 10**(Eb_N0_dB / 10)
num_symbols = 2000000
num_bits = num_symbols * 2
BER = np.zeros(len(Eb_N0_dB))

for i,eb_n0 in enumerate(Eb_N0):
    bits = np.random.randint(0,2,num_bits)

    mod = (bits[::2]*2-1) + 1j*(bits[1::2]*2-1)
    print(mod)

    N0 = 1/(eb_n0)
    noise  = np.sqrt(N0/2)*(np.random.randn(num_symbols) +
1j*np.random.randn(num_symbols))

    received = mod + noise
    demod_symbols = np.sign(np.real(received)) + np.sign(np.imag(received))*1j

    demod_bits = np.zeros(num_bits)
    demod_bits[::2] = (np.real(demod_symbols) > 0).astype(int)
    demod_bits[1::2] = (np.imag(demod_symbols) > 0).astype(int)

    BER[i] = np.sum(np.abs(bits - demod_bits)) / num_bits
```

2.1.3. Quadrature Amplitude Modulation (QAM)

QAM is a signal in which two carrier signals shifted in phase by 90° are modulated and combined, allowing it to encode multiple bits per symbol [9]. The common types of QAM include 16 , 32 , 64 , 128 and 256 QAM, where the coefficient of QAM represents the number of distinct states, and the root of the coefficient represents the numbers of bits encoded per symbol.

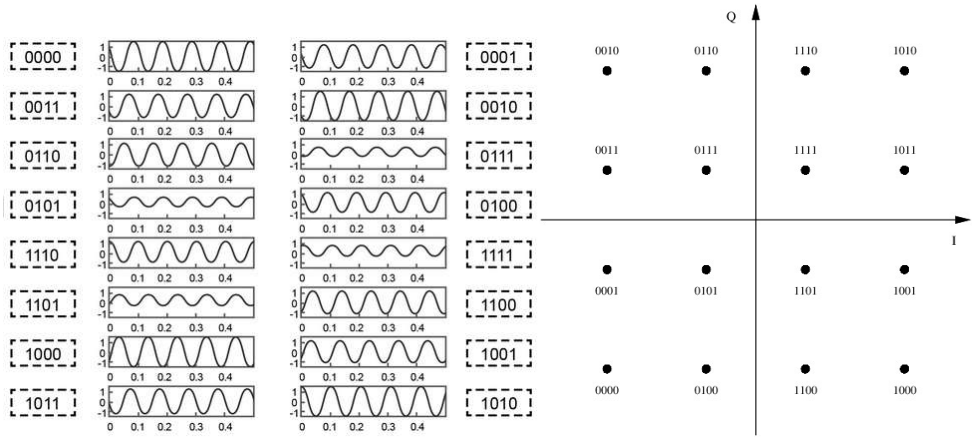

Fig. 5 & 6. Modulated Signals in the and their corresponding binary digits in 16-QAM [10] and the constellation diagram for 16-QAM [11]

16 - QAM in particular, as illustrated in Figures 5 and 6 above, represents 16 distinct states, encoding 4 bits per symbol. Each point on the constellation diagram has a coordinate, of which the value of the x coordinate is derived from the in-phase axis while the value of the y coordinate is derived from the quadrature axis. In order to generate the constellation for 16-QAM, 2 matrices are created, each 4x4, one representing the value of the x coordinates of the constellations and the other the value of the y coordinates of the constellations. They are then added together to form a matrix which represents the final constellation for 16-QAM. The modulation process involves utilising the decimal represented by the 4 bits to generate a symbol, which is the constellation that corresponds to the decimal on the constellation diagram.

For the process of demodulation, an IQ detection technique using minimum Euclidean distance was utilised [12]. The first step is to compute the pairwise Euclidean distance between the reference array and the received symbols corrupted with noise. Next, the symbols, from the reference array, that provide the minimum Euclidean distance between itself and the received symbol vector are returned. The Euclidean distance is calculated using the given formula:

$d(x,y) = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\ldots+(x_n-y_n)^2}$ where $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ are 2 points in n-dimensional space.

Finally, the symbols are demodulated back into the bits. The code that executes the entire process for 16 QAM is as follows:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
from scipy.special import erfc
import math
M = 16
```

```
def modulate(symbols):
    n = np.arange(0,M)

    #conversion of symbols to gray code
    gray = np.asarray([x^(x>>1) for x in n])

    #mapping onto constellation diagram
    dimensions = np.sqrt(M).astype(int) #dimensions of the map
    gray = np.reshape(gray,(dimensions,dimensions)) #creating the matrix
    gray[1::2] = np.fliplr(gray[1::2])
    g_walk = np.reshape(gray,(M))
    x = []
    y = []
    for i in range (dimensions):
        x += [i] * dimensions
        y += list(range(dimensions))
    x = np.array(x).astype(int)
    y = np.array(y).astype(int)
    Ax = 2*x+1-dimensions
    Ay = 2*y+1-dimensions
    global constellation
    constellation = Ax+1j*Ay
    mod = constellation[symbols]
    return mod

def demodulate(received):
    XA = np.column_stack((np.real(received),np.imag(received)))
    XB = np.column_stack((np.real(constellation),np.imag(constellation)))
    d = cdist(XA,XB,metric = 'euclidean')
    demod = np.argmin(d,axis=1)
    return demod
```

2.1.4. Additive White Gaussian Noise (AWGN) Channel

AWGN is a type of noise that is frequently used to model random disturbances in digital communication systems. It is usually modeled as a Gaussian distribution, also known as normal distribution. The code that is used to generate AWGN in the simulations is as follows:

```
def awgn(s,SNRdB,L=1):
    gamma = 10**(SNRdB/10)
    P=L*np.sum(np.sum(np.abs(s)**2))/len(s)
    N0=P/gamma
```

```
    noise = np.sqrt(N0/2)*(np.random.standard_normal(s.shape) +
1j*np.random.standard_normal(s.shape))
    received = s + noise
    return received
```

2.1.5. Encoding

2.1.5.1. Gray coding with k-maps

Gray coding is the ordering of bits whereby two adjacent symbols only differ by one bit in order to restrict the erroneous symbol decisions to single bit errors. This is usually achieved by converting the input symbols to Gray coded symbols and then mapping it to the desired QAM constellation.

2.1.5.2. Error correction code

Error correction code is a technique used in computers to identify and fix errors in data transmission which may arise as a result of noise and interference.

One of the more common types of error correction code is the Hamming code. It is used to detect and correct single-bit errors and involves the addition of parity bits in order to ensure that the total number of 1's in the data is even or odd [13]. The (7,4) Hamming Code in particular, involves the encoding of 4 bits of data into 7 bits by the addition of 3 parity bits.

$$G=\begin{bmatrix}1&0&0&0&1&0&1\\0&1&0&0&1&1&0\\0&0&1&0&1&1&1\\0&0&0&1&0&1&1\end{bmatrix}$$

Fig. 7. Generator Matrix for (7,4) Hamming code

In order to generate the bits to be transmitted, a generator matrix, G, as shown in figure 7 above is used. The generator matrix is multiplied by the 4 bits to be transmitted to form the 7 bit code that is transmitted. The generator matrix was also used to generate an array detailing the 16 possible permutations of binary strings that can be formed with the 7 bits. This matrix will be multiplied by the generator matrix to give the reference array.

The demodulated bits will then be compared with the reference array. The demodulated bits will be compared to every permutation within the reference array. The permutation that is the most similar to the demodulated bits will then be returned, which corrects whatever error that may have arisen as a result of noise or interference. The (7,4) Hamming Code is as follows:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erfc
```

```
#functions for hamming code encoding and decoding
gen_matrix = np.array([[1,1,1,0,0,0,0],[1,0,0,1,1,0,0],[0,1,0,1,0,1,0],[1,1,0,1,0,0,1]])

def hamming_code(data):
    hamming_code = np.zeros((7,), dtype = int)
    hamming_code = np.dot(data,gen_matrix)%2
    return hamming_code
#print(gen_matrix)

perm_array = np.array([list(map(int, bin(i)[2:].zfill(4))) for i in range(16)]) #permutations
of the different codes that can be formed w/ the 4 bits
#print(perm_array)

ref_array = np.dot(perm_array,gen_matrix)%2 #array of hamming codes
#print(ref_array)

def received_data(received):
    arr_diff = np.abs([array-received for array in ref_array])
    most_similar_index = np.argmin([np.sum(diff) for diff in arr_diff])
    most_similar_array = ref_array[most_similar_index]
    parity_pos = np.zeros(3, dtype = int)
    for i in range (3):
        parity_pos[i] = 2**i - 1
    received_data = np.delete(most_similar_array,parity_pos)
    return received_data
```

3. Results and Discussion

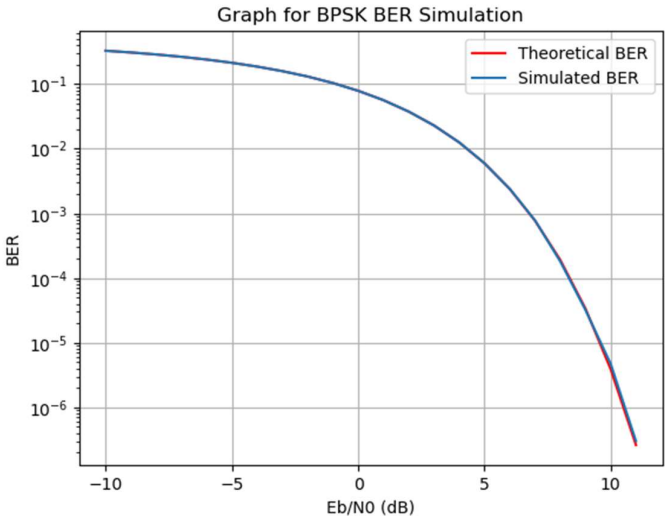

Fig. 8. Graph of the BER for BPSK across an AWGN Channel

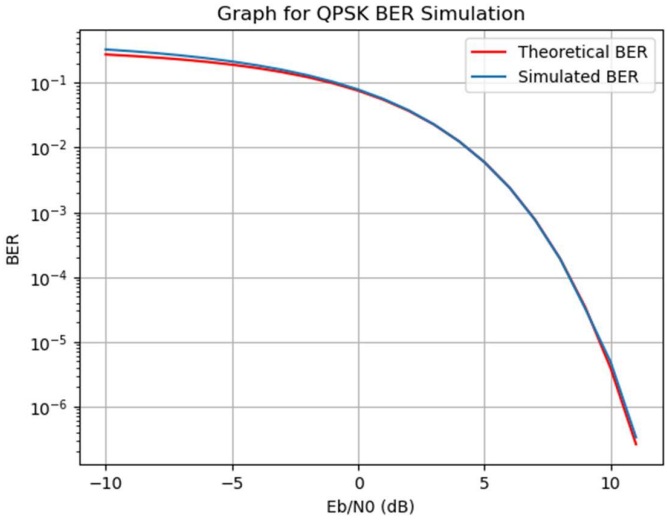

Fig. 9. Graph of the BER for QPSK across an AWGN Channel

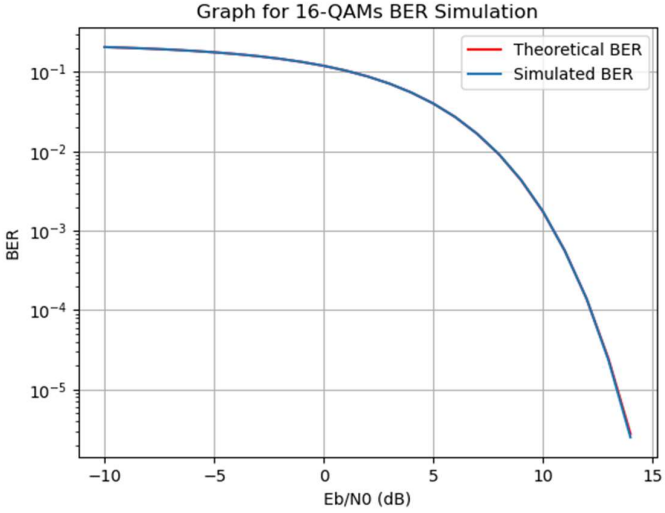

Fig. 10. Graph of the BER for 16 QAM across an AWGN Channel

Figures 8, 9 and 10 show the performance analysis of BPSK, QPSK and 16-QAM modulation schemes across an AWGN channel. The curves of the 3 modulation schemes are all decreasing at an increasing rate. As the energy per bit increases, the BER decreases, indicating that performance is inversely correlated to energy per bit. At an energy per bit to noise power spectral density ratio of 10, the BER for 16-QAM is the highest amongst the 3 while the BER for QPSK and BPSK are identical. This can be attributed to the fact that BER has been measured in terms of energy per bit, given by the formula $E_b = P \times T_b$. For BPSK works out to $E_b = P \times T_b$ while for QPSK, it is $E_b = \frac{2P \times T_b}{2}$ which works out to $E_b = P \times T_b$ as well. In sum, the distance between the constellation points are identical and hence, they are equally susceptible to noise, hence, they have an identical BER. However, QPSK is advantageous as it offers twice the bandwidth efficiency of BPSK [14]. The Bit Error Rate (BER) for 16 QAM is generally higher than the other modulation schemes because the constellation points are closer together, making it more susceptible to noise and interference, which leads to a higher probability of decoding errors [15].

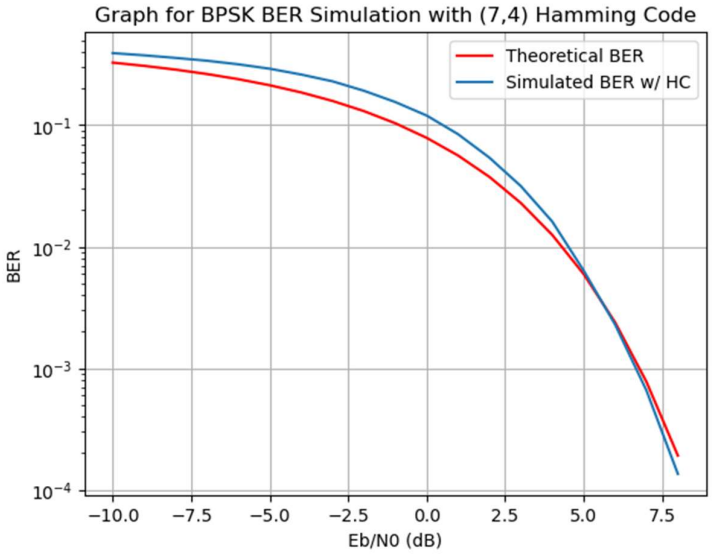

Fig. 11. Graph of the BER for BPSK with (7,4) Hamming Code across an AWGN Channel

The initial BER for BPSK with (7,4) Hamming Code was higher than the BER for BPSK. However, at a higher energy per bit, BPSK with Hamming Code starts to shine, displaying a lower BER than BPSK. The (7,4) Hamming Code introduces redundancy in the form of parity bits to enable error correction, which initially increases the BER because the system must handle additional bits.

Acknowledgements

I would like to express my gratitude for my mentor, Dr Tan Beng Soon, for his invaluable time and support throughout my time at DSO. I would also like to thank DSO for the invaluable opportunity offered to me and the resources catered to me during my time at DSO.

References

[1] Vikram Arkalgud Chandrasetty and Syed Mahfuzul Aziz, "Introduction," *Elsevier eBooks*, pp. 1–4, Jan. 2018, doi: https://doi.org/10.1016/b978-0-12-811255-7.00001-0.

[2] J. Walrand and P. Varaiya, "Wireless Networks," *High-Performance Communication Networks*, pp. 305–361, 2000, doi: https://doi.org/10.1016/b978-0-08-050803-0.50012-5.

[3] Ensatellite, "Binary Phase-Shift Keying (BPSK) – ensatellite," *Ensatellite.com*, Oct. 26, 2023. https://ensatellite.com/bpsk/

[4] N. Gao and S. Shimamoto, "Amplitude and Phase Modulation for Ultrasonic Wireless Communication," *International journal of wireless and mobile networks*, vol. 6, no. 2, pp. 01-12, Apr. 2014, doi: https://doi.org/10.5121/ijwmn.2014.6201.

[5] Mathuranathan, "Binary Phase Shift Keying (BPSK) – modulation and Demodulation – GaussianWaves," *gaussianwaves*, Apr. 08, 2010. https://www.gaussianwaves.com/2010/04/bpsk-modulation-and-demodulation-2/

[6] Ensatellite, "Quadrature Phase-Shift Keying (QPSK) – ensatellite," *ensatellite*. https://ensatellite.com/qpsk/

[7] L. A. Camuñas-Mesa and J. M. de la Rosa, "Combining Software-Defined Radio Learning Modules and Neural Networks for Teaching Communication Systems Courses," *Information*, vol. 14, no. 11, pp. 599–599, Nov. 2023, doi: https://doi.org/10.3390/info14110599.

[8] R. Keim, "Understanding Quadrature Phase Shift Keying (QPSK) Modulation - Technical Articles," *www.allaboutcircuits.com*, Aug. 17, 2016. https://www.allaboutcircuits.com/technical-articles/quadrature-phase-shift-keying-qpsk-modulation/

[9] I. Poole, "What is QAM: Quadrature Amplitude Modulation» Electronics Notes," *www.electronics-notes.com*. https://www.electronics-notes.com/articles/radio/modulation/quadrature-amplitude-modulation-what-is-qam-basics.php#google_vignette

[10] RF, "What is 16-QAM Modulation? - everything RF," *Everythingrf.com*, Jul. 19, 2022. https://www.everythingrf.com/community/what-is-16-qam-modulation

[11] T. T. Kim, D. Prakash, E. Bragnell, P. Kjellander, and A. Nezirovic, "MIMO Multiple Input Multiple Output (MIMO) Smart Antenna Over Radio," *ResearchGate*, Jun. 02, 2003. https://www.researchgate.net/publication/264874518_MIMO_Multiple_Input_Multiple_Output_MIMO_Smart_Antenna_Over_Radio

[12] M. Viswanathan, *Digital Modulations Using Python*. 2019.

[13] H. Pandey, "Hamming Code in Computer Network - GeeksforGeeks," *GeeksforGeeks*, Dec. 26, 2017. https://www.geeksforgeeks.org/hamming-code-in-computer-network/

[14] T. J. Lim, L. K. Rasmussen, and H. Sugimoto, "Relative Performance of BPSK and QPSK in the Presence of Complex Multiuser CDMA Interference," *Wireless Personal Communications*, vol. 13, no. 3, pp. 237–256, 2000, doi: https://doi.org/10.1023/a:1008974119760.

[15] K. Sankar, "Comparing 16PSK vs 16QAM for symbol error rate," *dsplog*, Sep. 15, 2012. https://dsplog.com/2008/03/29/comparing-16psk-vs-16qam-for-symbol-error-rate/#google_vignette