Simulation of a Digital Communication System to Assess the BER in Noisy Channels Using Arduino

Oscar Carrion

October 28, 2024

Simulation of a digital communication system to assess the BER in noisy channels using Arduino

Oscar Carrion
Dept. of Telecommunications
Pontificia Universidad Catolica del Peru
Lima, Peru
https://orcid.org/0009-0005-8041-8849

***Abstract*—In this work, a digital baseband communication system is simulated using the Arduino environment and a C language program. In this program we can adjust several parameters of the transmission system, such as the number of data bits and the noise level in the channel. Then, this C program is used to simulate a noisy channel and calculates the number of error bits during transmission essays. This simulation will serve to calculate and analyze the BER (bit error rate) vs SNR (signal to noise ratio) ratio which is a standard for quality assessments in a digital communications system. In addition, Hamming coding techniques are implemented in the system to allow a comparison between a transmission with and without channel coding.**

Keywords—Digital communication, BER vs SNR, noisy channel.

I. INTRODUCTION

Simulating a digital communication system is important since it allows us to evaluate the quality of communication when digital information is transmitted over a noisy channel [1]. Additionally, implementing a simulator enables students and researchers to observe how noise affects transmitted information and experimentally calculate BER vs. SNR curves [2].

Using the Arduino module for simulation has several advantages: it encourages creative thinking and imagination in students, serves as a low-cost platform for creating projects (simulations), and features a C programming language that is easy to learn and understand, making Arduino programs easy to handle and improve [3].

Nowadays, there are several simulators of digital communication systems. Most of them related to this paper are quoted following: In [4], a study about the significance of information theory is presented and then, channel coding techniques are explained with a number of simulations. Most of these simulations are performed by MATLAB. On the other hand, paper [5] presents a method to modify the PDF (probability distribution function) where non-linearity in the channel is considered. Once the PDF is correctly modified, the tail probability methods to determine BER can be applied. Other simulation program is described in [6] where a complete communication system model is simulated including source coding, channel coding, digital modulation and the various types of antenna array systems. Also a AWGN (Additive White Gaussian Noise) channel is implemented in the simulation program.

The remainder of this work is structured as follows: Chapter 2 describes a mathematical model of the digital communication system, implemented through various functions in a C program. Chapter 3 details the experimental calculation of the BER using the proposed simulator, along with graphical results such as BER vs SNR curves and time-domain representations of transmitted and received signals. Finally, in Chapter 4 conclusions are addressed.

II. SYSTEM DESIGN

Here, the parts of the designed communication system are described and how they were implemented in the Arduino program.

A. *Digital Data Generator*

The digital data generator generates digital data in blocks of 4 bits. These 4 bit-blocks will be suitable to perform the channel coding later. The 4 bit-blocks are obtained using (1).

$$\begin{aligned} &s(\theta,k)=[\![A\times\sin(\theta)+B]\!]\\ &A=2^{(k-1)}-1\\ &B=2^{(k-1)} \end{aligned} \qquad (1)$$

Where A, B $\in \mathbb{Z}^+$, k is the length of bit blocks and $\theta \in \mathbb{R}$ is the angle of the sine function in radians. Equation (1) is obtained experimentally and yields integer sinusoidal data.

We use sinusoidal data in the communication system because it is easily recognizable at the receiver and simple to generate [7], [8]. We need only positive values which are obtained by adding an offset B to the sine function (1). Finally, the greatest integer function $[\![\;]\!]$ rounds the values resulting in positive integer values between B – A and B + A.

If we replace k = 4 into (1), we get (2):

$$s(\theta,4)=[\![7\times\sin(\theta)+8]\!] \qquad (2)$$

The output in (2) is in the range from 1 to 15. In base-2, from $0001_{(2)}$ to $1111_{(2)}$.

Equation (2) is programmed into the *gen.sine* function, as shown in (3). This function has two arguments: Δt is the time interval between two consecutive data and T is the number of periods of the sine wave. Both arguments determine the amount of data to transmit.

$$data=gen.sine(\Delta t,T) \qquad (3)$$

The function (3) is programmed in the microcontroller embedded in an Arduino Nano board according (2).

The amount of data required, named N_B, can be estimated using the confidence level formula [9], given by (4).

$$N_B = \frac{-\ln(1-CL)}{BER_E} \quad (4)$$

Where CL is the percentage confidence level and BER_E is the expected error rate for a given application. For the current application we have CL = 95% [9] and $BER_E = 10^{-3}$ if we consider transmitting voice signals. When those values are substituted in (4), we need $N_B \geq 3000$ bits of data to experimentally calculate the BER.

B. *Hamming 7,4 Encoder*

A Hamming encoder with a binary multiplier is implemented, which multiplies the data generated in (3) by the parity matrix in (5) and then the parity bits are obtained. The parity bits are then joined to the left of the message (data) using the left shift operator (<<), as done in (6) [10].

$$parity = data \times \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad (5)$$

$$code = (parity \ll 4) + data \quad (6)$$

Operations in (5) and (6) are implemented in function (7) where the encoding algorithm is performed. This function receives data with a length of 4 bits as an argument and returns data with a length of 7 bits, which are stored in the variable "*code*" given in (7). This variable represents the encoded word [10].

$$code = Hamming.enc(data) \quad (7)$$

Next, the code word in (7) is send through the channel corrupted by noise. The generation of noise will be explained in the following section.

C. *Generation of noise*

Noise is necessary to simulate the transmission channel of digital communication system, so a logical noise generator will be implemented in this section.

Logical noise is defined by 2 values: "0" indicates that there is no errors and "1" indicates that one error exists. The logic noise generator must generate error noise blocks of 4 or 7 bits in length [10].

Noise blocks are generated using the *gen.noise* function, as in (8). The arguments of (8) are N_0 and L_b. N_0 is the percentage of error blocks from the whole data in a test transmission. L_b defines the size of the noise block and can take 2 values: If L_b is 2^4, it will generate 4-bit noise blocks and if L_b is 2^7 it will generate 7-bit noise blocks. Variable N_0 is equivalent to the noise intensity levels at this model.

$$W_i = gen.noise(N_0, L_b) \quad (8)$$

The function (8) generates integer random blocks within the range $[0, 2^{L_b} - 1]$. During a test transmission, depending on the error percentage (N_0), some blocks (W_i) will be error-free, while the remaining blocks will contain between 1 to 3 errors. The remainder blocks with errors must be programmed with a Gaussian PDF [11]. This PDF is a standard distribution for modeling noise in digital communication channels [11]. The *Gaussian* distribution is defined in (9) [12].

$$P_e(x) = \frac{1}{\sqrt{2\pi\sigma}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (9)$$

Simplifying (9) with mean equal 0 and variance equal 1 we obtain the normalized Gaussian distribution. A graphical approximation is shown in figure 1 [12].

To simplify the channel model, we can limit the number of errors in each noise block generated from 1 to 3 bits. Errors greater than 3 bits are neglected.

Now, we can divide the Gaussian PDF in 3 regions as shown in Figure 2 [12]. Region I has an error probability for blocks with 1-bit errors. Similarly, regions II and III correspond to error probabilities for blocks with 2-bit and 3-bit errors, respectively.

According Figure 2, we can calculate the error probability for blocks with 1-bit error in (10) [12]:

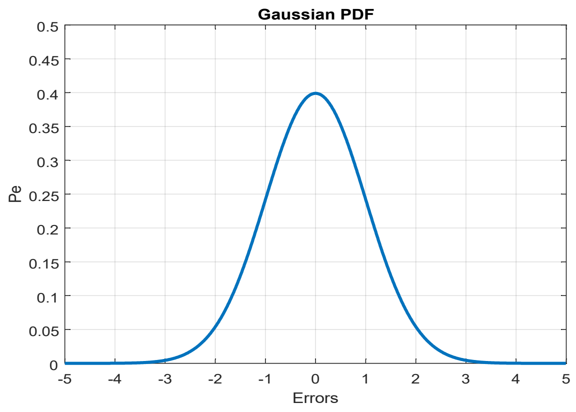

Fig. 1: Normalized Gaussian distribution

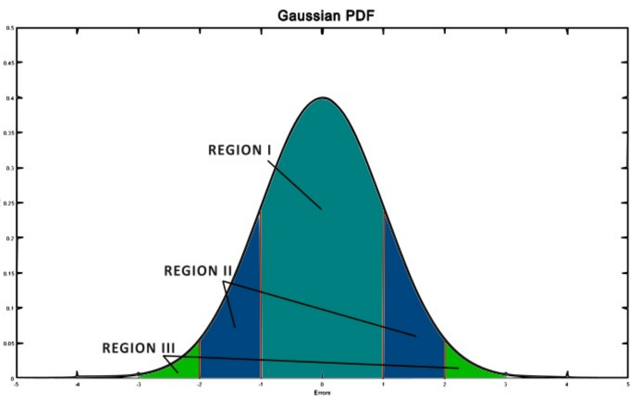

Fig. 2. Probability regions of errors

$$P_e(x=1)=\frac{1}{\sqrt{2\pi}}\int_{-1}^{1} e^{-x^2/2}dx \quad (10)$$

Solving (10) from tables [12] yield Pe(x=1) = 0.684. Similarly, we calculate the Pe(x=2) with integration limits in region II and Pe(x=3) with integration limits in region 3. Results are Pe(x=2) = 0.273 and Pe(x=3) = 0.0426.

Once the error probabilities for the 3 cases of error blocks have been defined, an algorithm is configured in (8), whose explanation is detailed below:

- First, we use the Random() function, which follows a uniform PDF [14]. This function generates two random numbers: n_1, which takes values from 1 to 100, and n_2, which is a noise block with a uniform PDF, but it must have a Gaussian PDF [12]. To carry the conversion out [13], we count the bits in n_2 using a specific bit-counting algorithm [15]. The result of the counting is stored in the variable n_b..

- Next, two probability decision thresholds 'Th1' and 'Th2' (Th1 > Th2) are defined to classify the error blocks. Thresholds are determined using the average value between 2 previously calculated error probabilities Pe(x). For example, for Th1:

$$Th1=\frac{P_e(x=1)+P_e(x=2)}{2} \quad (11)$$

Similarly for Th2. Numeric values are Th1 = 0.478 and Th2 = 0.157. Thus, following rules are made to classify error blocks.

- If n_1 > 100×Th1, all error blocks will be converted to 1-bit error blocks and flag = 1.
- If 100×Th2 < n_1 < 100×Th1, all error blocks will be converted to 2-bit error blocks and flag = 2.
- If n_1 < 100×Th2, all error blocks will be converted to 3-bit error blocks and flag = 3.

- We find the difference between n_b and the 'flag' such as (12).

$$n_{diff}=n_b-flag \quad (12)$$

- Following, we use this difference to convert error blocks using an algorithm [15] to add or remove complementary bits in error blocks, according the next rules:

- If $n_{diff}>0$, remove a bit in n_2 until $n_{diff}=0$.
- If $n_{diff}<0$, add a bit in n_2 until $n_{diff}=0$.
- if $n_{diff}=0$ return n_2. Then W_i = n_2 in (8).

After generating noise error blocks using (8) and storing them in 'W_i', these blocks are added to the 'data' variable in (3) for uncoded test transmission, or to the 'code' variable in (7) for test transmission with Hamming coding. The XOR operator is used to add noise blocks because it does not produce a carry, as in (13).

$$\begin{aligned} R_d&=data\oplus W_4 \\ R_c&=code\oplus W_7 \end{aligned} \quad (13)$$

D. *Hamming 7,4 Decoder*

A Hamming decoder is implemented with a binary multiplier (as in the Hamming encoder) that multiplies the R_C block in (13) by a parity check matrix and the syndrome vector S_n is obtained in (14) [10]. The syndrome vector is used to find and correct 1-bit error in the variable 'R_C' given in (13). Function f(S_n) in (15) represents the standard array used to find the position of wrong bits in noise blocks [10].

$$S_n=R_c\times\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\1&1&0\\0&1&1\\1&1&1\\1&0&1\end{bmatrix} \quad (14)$$

$$decode=R_c\oplus 2^{f(S_n)} \quad (15)$$

Operations in (14) and (15) are implemented in function (16) in the simulation program. This function receives coded data plus logical noise from (13), and return the variable "*decode*".

$$decode=Hamming.dec(R_c) \quad (16)$$

Finally, the original data can be recovered trimmed the decode variable from (16) using the AND operator as in (17).

$$rec.data=(decode\wedge 0x0F) \quad (17)$$

III. Simulation Results

At this point, two digital communication systems are simulated as shown in Figure 3. Both systems have the same data source defined in Section II-A. The difference between the systems is that the lower system has Hamming coding implemented. At the receiver, bit errors are identified by comparing the original data with received data corrupted by logic noise.

First, a quantitative procedure is carried out, where the transmitted bits and error bits are counted using the created program and, from this information, the BER for various noise levels N_0 is calculated. On the other hand, a qualitative procedure is carried out, where the original data and the received data are plotted for both systems shown in Fig. 3. In this way, the impact of noise in a digital transmission can be evaluated in the time domain.

A. *Quantitative Results*

The value of BER for various noise levels can be calculated using (18). The input arguments are obtained directly from the data printed by the Arduino Nano board to the serial monitor of the computer.

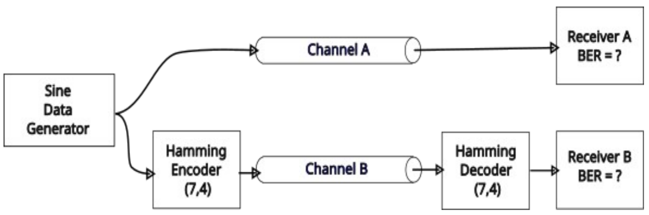

Fig. 3. Transmission scheme to be simulated

$$BER=\frac{\sum_{i=1}^{3}(w_i \times wrong\,data)}{Total\,data \times (N°\,bits/data)} \qquad (18)$$

Where w_i is the amount of bit errors in a wrong datum. The BER is calculated using (18) for both cases: System without coding and System with Hamming coding. Eleven noise levels and N_B = 7406 data bits at most according (4) are taken for the calculation. An example is given below for a noise level of 15%, 503 data samples, and 7 bits per sample.

- N° Wrong bits = (1)58+(2)11+(3)13 = 119
- Total data = 503
- N° Total bits = 503(7 bit) = 3521

Then, the BER in this case is:

$$BER=\frac{119}{503\times 7}=0.01732$$

The BER calculation for other noise levels and system types are tabulated in Table I.

In Fig. 4, the collected data is represented through the BER vs SNR curves. In the figure, the units of the SNR axis are in decibels (dB) and the BER axis should be on a logarithmic scale for a standard representation of BER, such as in reference [11]. To transform noise levels (N_0) to dB we use the following formula [11]:

$$SNR(dB)=10\log(1/N_0) \qquad (19)$$

The BER vs SNR curves in Fig. 4 are very similar to the curves in references [11], [16]-[18]. In this figure, we observe the red curve, which are the encoded data, has a lower BER than the green curve which represents the unencoded data and, on the other hand, we observe that the lower the noise level, the lower the BER.

B. *Qualitative Results*

The original sine signal from (3) and the received signal from both systems given in Fig. 3 are plotted simultaneously in real time using a script of MATLAB.

The script reads data from 3 sources, which are multiplexed in time. The sources are: the original sinusoidal signal, the received sinusoidal signal without coding (upper system in fig. 3), and the received sinusoidal signal with coding (lower system in fig. 3).

The script uses the *readline()* function that belongs to the "USB and serial communication" library of MATLAB [19]. This function allows to read and demultiplex the data,

TABLE I. CALCULATION OF BER FROM SIMULATION

	BER	
% error levels N_0	***Without coding***	***With Hamming coding***
0.05	0.01870	0.00446
0.1	0.03741	0.01272
0.2	0.07978	0.02056
0.3	0.12098	0.03450
0.4	0.14962	0.06791
0.5	0.17116	0.06723
0.6	0.21330	0.07210
0.7	0.28220	0.08577
0.8	0.26918	0.07806
0.9	0.27841	0.07860
1.0	0.35890	0.08739

then this separated data is saved in the "*data*" array, according to (20).

$$\text{data(M,p) = readline(S\_arduino)} \qquad (20)$$

Where ‘M’ is the integer number of data and p is the selection variable of the multiplexed data. The variable 'p' can take 3 values depending on the source of data read. The argument ‘S_arduino’ in (20) is an object that stores the parameters of the serial communication, which are the port name and baud rate.

Once the data was read and separated, they are plotted in Fig. 5. The signals in Fig. 5 are described as follows: the blue signal is the original sine wave signal before being transmitted, the red signal is the received sine wave signal without coding, and the magenta signal is the received sine wave signal with the effects of the Hamming 7,4 coding.

A comparison between the signals is made in Fig. 5. There, we can see that the red signal without coding has more data errors than the magenta signal with Hamming 7,4 coding, which also has a lower BER, according to Fig. 4.

Therefore, it has been experimentally proven that channel coding reduces data errors in a digital transmission.

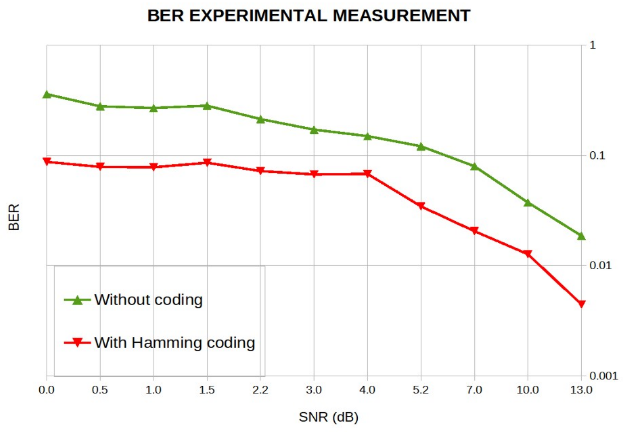

Fig. 4. BER vs SNR for transmitted data.

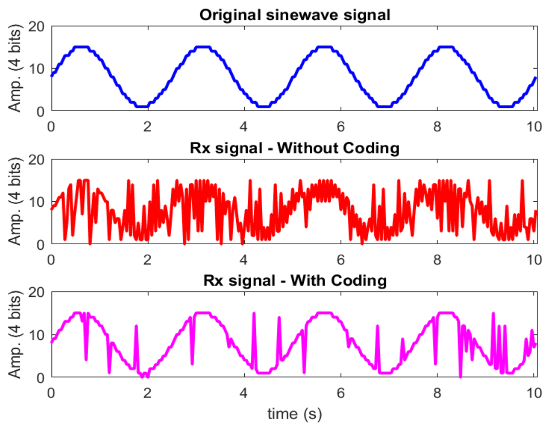

Fig. 5. Comparison of the transmitted signals.

IV. Conclusions

The main contribution of this paper is the simulation developed entirely within the Arduino environment. This simulation is valuable because digital transmissions are tested on external hardware, with signals represented by voltage levels that can be transmitted over physical guided media. Therefore, the simulation closely mirrors real-world systems. Furthermore, the simulated system can help development of creativity and technical abilities for undergraduate students and researches.

During the simulations carried out in section III, a total of 1058 4-bit and 7-bit blocks were used. Since this is a statistical process, the measured BER only approaches the actual BER when the number of tested bits approaches infinity [9]. Therefore, the measured BER will be closer to the actual BER if the number of bit samples increases.

The noise model defined in section II-C can be used in other applications or digital communication systems where discrete noise is required to analyze such systems. These applications can include channel coding, source coding, channel equalization, among others.

References

[1] M. C. Jeruchim, P. Balaban, K S. Shanmugan, Simulation of Communication Systems Modeling, Methodology and Techniques. 2nd ed., 2000, pp 1-12.
[2] J. G. Proakis, M. Salehi, G. Bauch, Contemporary Communication Systems Using MATLAB, 3th. ed., 2013, p. iii.
[3] J. A. Tukhtanazarovich, "Advantage of the Arduino Platform In Forming Creative Skills In Youth" JournalNX- A Multidisciplinary Peer Reviewed Journal, July 2021.
[4] G. Sheng, X. Zhao, H. Zhang, H. Song, Z. Lv., "Mathematical Models for Simulating Coded Digital Communication: A Comprehensive Tutorial by Big Data Analytics in Cyber-Physical Systems". IEEE 2016.
[5] G. Malhotra, "Method for analytically calculating BER in presence of non-linearity", Xilinx, DesignCon 2014.
[6] A. Güngör, F.Arıkan, O. Arıkan, "Simulation of a Digital Communication System", Ankara, Turkey.
[7] V. Papez, J. Roztocil and S. Dado, "Sine wave signal sources for testing high-speed High-resolution a/d converters," XIX IMEKO World Congress, January 2009.
[8] B. K. Vasan, S. K Sudani, D. J Chen, R. L Geiger, "Sinusoidal signal generation for production testing and BIST applications," Conference Paper, May 2012.
[9] N. Faubert, "BER – Is it Bit Error Rate or Bit Error Ratio?", © Keysight Technologies 2000–2024, March 2019.
[10] S. Haykin, Communication Systems, 4th ed., 2001, pp. 3, 626-641.
[11] L. W. Couch, Digital and Analog Communication Systems, 8th ed., 2013, pp. 20-28, 682-703.
[12] S. Lipschutz, M. Lipson, Schaum's Outline of Probability, 3rd ed., 2021. pp. 105-108.
[13] G. Roussas, An Introduction to Probability and Statistical Inference, 2nd ed., 2015, pp. 207-243.
[14] Random Numbers, Arduino Documentation-Programming, © Arduino 2024.
[15] B. Kernighan, D. M. Ritchie, The C programming Language, 2nd ed., 1988, p. 50.
[16] W. Rurik, A. Mazumdar, "Hamming Codes as Error-Reducing Codes," 2016 IEEE Information Theory Workshop (ITW), September 2016.
[17] Nasaruddin, B.Yuhanda, Elizar, Syahrial, "Design and performance analysis of channel coding scheme based on multiplication by alphabet-9", Journal of Telecommunication, Electronic and Computer Engineering.
[18] R. E. Ziemer, W. H. Tranter, Principles of Communication, 7th ed., 2015. p. 652.
[19] Serial Port Devices Functions, MATLAB Documentation, © 1994-2021 The MathWorks, Inc.