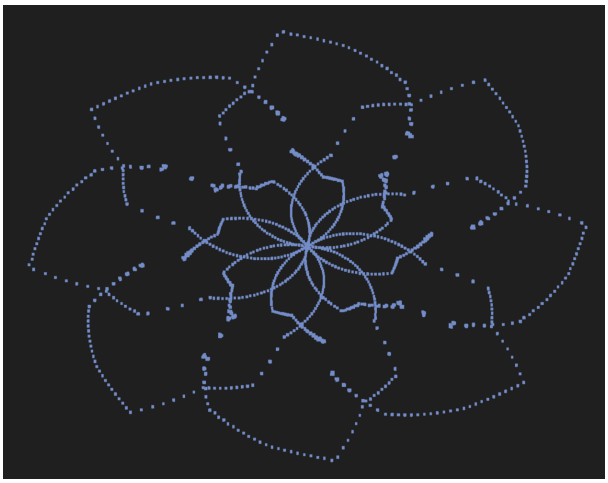
OFDM Project Report

École Polytechnique Fédérale de Lausanne



EE-442 : Wireless receivers algorithms and architectures Project

Ennouri Hsan, Fleury Gael
Group 7
December 2025

Abstract

This report presents the design, implementation and analysis of an acoustic **Orthogonal Frequency Division Multiplexing (OFDM)** transmission system using MATLAB. The system includes frame synchronization, channel estimation, equalization, and phase tracking. Performance was evaluated across a variety of channel conditions, including the provided emulated channels as well as real acoustic transmissions.

Experimental results showed that the system achieves a Bit Error Rate (BER) of 0% in high-SNR acoustic environments and maintains reliable performance under moderate phase drift. The phase tracking, based on the Viterbi–Viterbi algorithm can significantly extend the number of correctly decoded OFDM symbols, though its effectiveness is limited when the phase variation exceeds $\pm\frac{\pi}{4}$.

The system was extended to include text / image transmission, a robust communication protocol with CRC-8 checks, and higher order M-QAM modulation. These additions significantly improve the system's throughput, flexibility, and practical utility.

Contents

System design and implementation

Figure 1.1 shows the global architecture of the system. It is composed of a **transmitter**, a **wireless channel** and a **receiver**. The specific architecture of these components is detailed below.

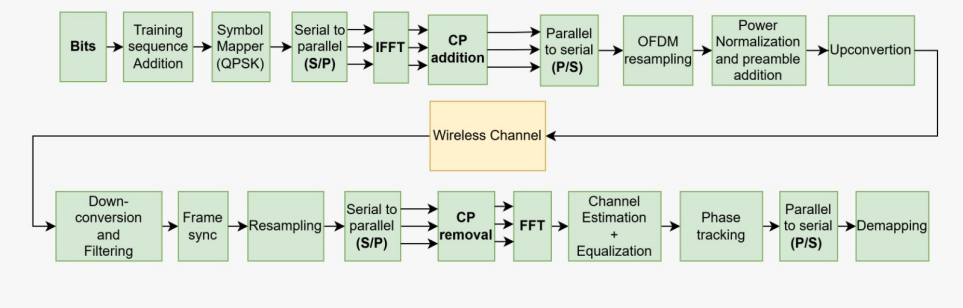


Figure 1.1: Block diagram of OFDM system

1.1 Transmitter

The transmitter has the following architecture :

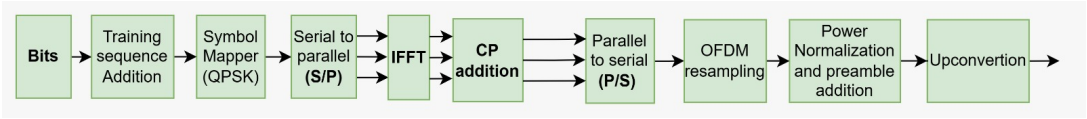
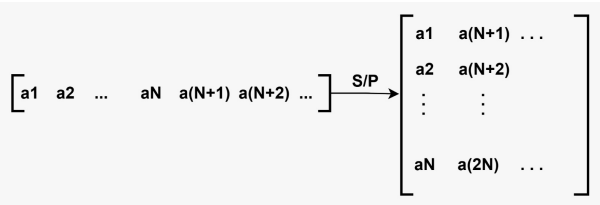
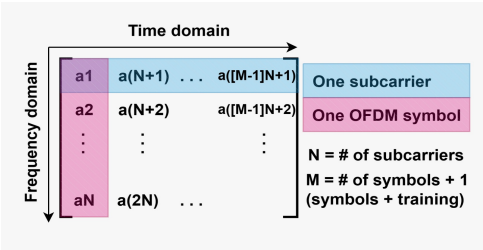


Figure 1.2: Transmitter architecture

1. **Data Generation:** Data is generated either as a stream of (pseudo) random bits or as a bitstream derived from an image file or from a typed text.
2. **Training sequence insertion :** A training sequence composed of $2N$ LFSR generated bits is inserted before the data bits. So the total number of data sent (per frame, excluding the preamble) is $\mathbf{M} = \text{numSymbols} + \mathbf{1}$ (Number of "useful" symbols per frame + 1 training sequence).
3. **QPSK mapping :** The bit sequence is mapped into QPSK symbols(**QPSK_map()** function).
4. **Serial to Parallel (S/P) :** The row vector containing **N*M** symbols is transformed into a (N x M) data matrix as shown in figure 1.3a(using the **serial_to_parallel()** function).
5. **IFFT :** An Inverse Fast Fourier Transform (IFFT) is applied to transform the data from the frequency domain to the time domain

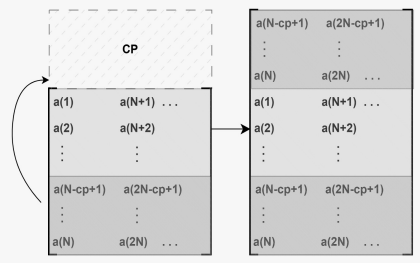


(a) Serial to Parallel

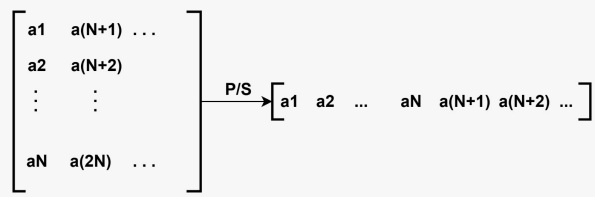


(b) One data frame architecture

6. **Cyclic Prefix (CP) addition :** A Cyclic Prefix (CP) is prepended to each OFDM symbol as shown in figure 1.4a(using the **add_cp()** function).



(a) CP addition



(b) Parallel to Serial

7. **Parallel to Serial (P/S) :** The data matrix is transformed back into a row vector.
8. **Preamble Generation :** A preamble is generated as a LFSR sequence and mapped into BPSK symbols. The preamble is then upsampled to 1kHz and pulse shaped using the **rrc()** function.
9. **OFDM resampling :** The OFDM symbols are resampled to the sound card frequency($f_s' = 48$kHz) using the **ofdm_tx_resampling()**
10. **Power normalization :** The power of both the preamble and the OFDM symbols is normalized and then scaled down to avoid clipping during audio transmission which would lead to degraded transmission performance.
11. **Preamble addition :** The single carrier preamble is added before the data. The final data architecture looks like below (Figure 1.5)

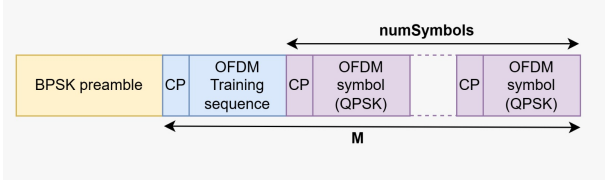


Figure 1.5: Data frame architecture

12. **Upconversion :** The signal is upconverted to the carrier frequency of the audio signal ($8kHz$). The signal is finally real-valued to be able to undergo transmission through speakers.

Key Remarks on the Transmitter:

- **Separate Processing Chains:** The preamble (Single Carrier) and OFDM data follow different processing paths (pulse shaping vs. no pulse shaping) and are only combined at the final stage.

- **Training Sequence:** The training symbols are generated using the same LFSR sequence as the preamble (via **lfsr_framesync()**).
- **Signal normalization:** Both the preamble and the OFDM symbols are normalized and then scaled down to avoid clipping during audio transmission. Clipping would otherwise distort the signal by cutting its peaks, leading to degraded transmission performance. Clipping may still occur due to the high peak-to-average power ratio (PAPR) of OFDM signals, which can produce large amplitude peaks when multiple subcarriers add constructively.
- **Resampling and Upsampling:** The preamble and the OFDM symbols are resampled separately before being stitched together. The preamble is set to a 1000 Hz symbol rate. The OFDM symbols are treated by the **ofdm_tx_resampling()**. Then the concatenated signals are upconverted to the carrier frequency. We need to be careful at this step to take into account the sampling of the audio card (at a frequency of $f_s' = 48kHz$). The signal is finally real-valued to be transmission through speakers, because sound waves are real physical pressure variations and cannot represent complex-valued polarizations, unlike electromagnetic waves can.

1.2 Channel

The signal is transmitted using a speaker, travels through a channel that is determined by the environment and will be captured by a microphone placed at a certain distance. The signal undergoes scattering, multipath propagation (reverberation), and phase shifts due to propagation delay. We were also provided with 5 "emulated" channels (ID 1-5), each one with its own specifications.

1.3 Receiver

The receiver has the following architecture :

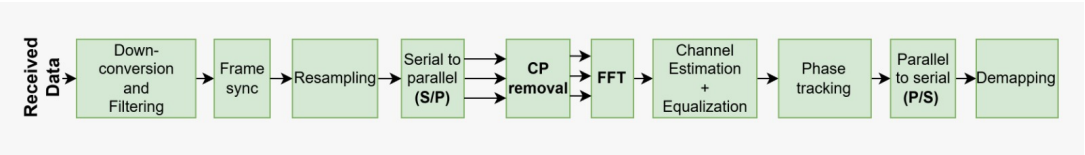


Figure 1.6: Receiver architecture

1. **Downconversion & Filtering :** The received signal is downconverted from the carrier frequency to the baseband. A low-pass filter (ofdmlowpass) is applied to keep only the useful signal.
2. **Frame Synchronization :** We locate the start of the frame using the **frame_sync()** function, which correlates the received signal with the known preamble sequence.
3. **Resampling :** The synchronized signal is resampled from the sound card rate f_s' back to the signal rate f_s using the **ofdm_rx_resampling()** function.
4. **Serial to Parallel (S/P) :** The signal is reshaped from a row vector into a matrix into vectors of $N + N_{cp}$.
5. **CP removal :** The **Cyclic prefix** is removed by discarding the N_{cp} samples of each symbol, leaving N useful samples. (See figure 1.7)
6. **FFT :** The Fast Fourier Transform is applied to convert the time-domain samples back to the frequency domain.

7. **Channel estimation :** The received training symbol is compared against the known training symbol to estimate the Channel Transfer Function $H[m]$
8. **Equalization :** The effects of the channel are reversed by dividing the received data symbols by the estimated channel response : $\hat{a}[n,m] = \frac{z[n,m]}{\hat{H}[n,m]}$

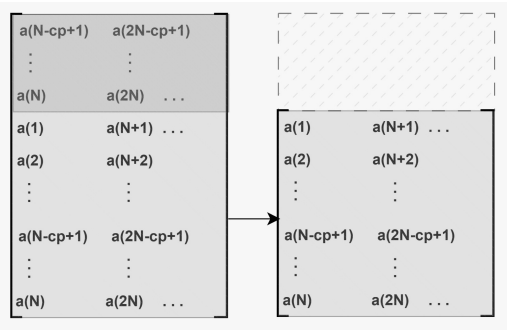


Figure 1.7: CP removal

9. **Phase Tracking/Correction :** A phase tracking algorithm is applied across the different subcarriers. This allows to correct the residual phase rotation of the signal.
10. **Parallel to serial (P/S) :** The data matrix is transformed back into a row vector.
11. **Demapping :** The complex symbols are mapped back to bits based on the QPSK constellation.

Key Remarks on the Receiver:

- **Separate Processing Chains :** As in the transmission block, here the OFDM symbols and preamble were handled separately, specifically the OFDM symbols were not convoluted with the **rrc()** function to keep them intact.
- **Filter Bandwidth:** The low-pass filter cut-off must be carefully selected slightly above the signal bandwidth to avoid attenuating the outer subcarriers (which would destroy the orthogonality)
- We had some problems at first because the preamble was resampled, which was destroying it and made the synchronization not possible. This issue was later fixed.
- One of the difficulties we encountered was to clearly understand the implementation of the **fft/ifft** functions in MATLAB and how the frequencies are mapped. With time we got a better understanding of how it is handled.

1.4 Explanation of the different functions

1.4.1 Channel Estimation

The channel estimation method uses a **Block-Type Pilot Arrangement** (also known as vertical estimation). A training, known sequence (Generated as an LFSR sequence) is sent as an OFDM symbol in the beginning of each frame. The estimation is performed as follows:

$$\hat{H} = \frac{Y_{train}}{X_{train}} \tag{1.1}$$

Subsequently, the payload data within each frame is equalized using the Zero-Forcing (ZF) criterion:

$$\hat{a}[n,m] = \frac{z[n,m]}{\hat{H}[n,m]} \tag{1.2}$$

Remark : The number of data symbols transmitted between training sequences (**nb_symbols**) can be tuned depending on the coherence time of the channel.

1.4.2 Phase Tracking: Viterbi-Viterbi

The current phase tracking algorithm is the **Viterbi-Viterbi** algorithm. This is a blind (non-data-aided) estimation method that works by raising the received signal to the M-th power (specifically the 4th power for QPSK) to eliminate the data modulation. The instantaneous phase offset is estimated as follows:

$$\hat{\theta}_{inst}[n] = \frac{1}{4} \arg\{-z[n]^4\} \tag{1.3}$$

This estimate is then smoothed using a recursive filter to obtain the final phase correction:

$$\hat{\theta}[n+1] = \alpha \cdot \hat{\theta}_{inst}[n] + (1-\alpha) \cdot \hat{\theta}[n] \tag{1.4}$$

Where the α term is a **smoothing factor** (or forgetting factor), chosen between 0 and 1. It controls the memory of the filter: a low α provides better noise rejection (smoothing) but slower reaction to phase changes, while a high α allows for faster tracking but is more sensitive to noise. This parameter can be tuned to provide better results.

1.4.3 CRC Calculation

For the version of the code that includes the communication protocol (see section 2.8), a **Cyclic Redundancy Check (CRC)** is used to ensure the integrity of the received header at the receiver. It consists of a mathematical operation performed on the (ID + number of data send) bits and whose result is stored in an 8 bit slot in the header.
The specific algorithm implemented is the **CRC-8-CCITT**. This method treats the data bits as a long binary number and divides it by a specific 9-bit pattern (the generator). The generator used is based on the polynomial $x^8 + x^2 + x + 1$, which translates to the binary sequence 100000111_2.
The process follows these three steps:

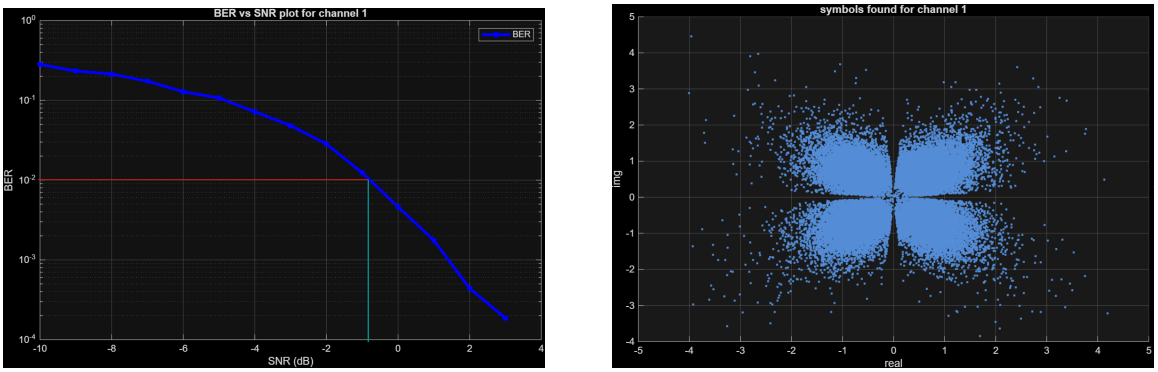
1. **Padding:** Eight zeros are appended to the end of the input data.
2. **Division:** The padded data is divided by the generator sequence (100000111_2) using bitwise XOR.
3. **Remainder:** The division produces a remainder of exactly 8 bits. This remainder is the CRC checksum.

At the receiver side, the same calculation is performed on the received header bits. If the locally computed CRC matches the received CRC field, the header is assumed to be uncorrupted, if not, an error is raised and the program stops.

Questions and extended system

2.1 Implementation of an OFDM transmission system

Figure 1.1 shows the block diagram of our OFDM system.
The system was tested with the channel 1 (AWGN channel) and a BER of 0 was achieved when setting the SNR to 100.
Figure 2.1a shows that a BER vs SNR plot for our OFDM system using the channel 1. We can see that for a $SNR > -1\,dB$, our system performs with a $BER < 1\%$. Figure 2.1b shows the received symbols while setting the SNR to -1.



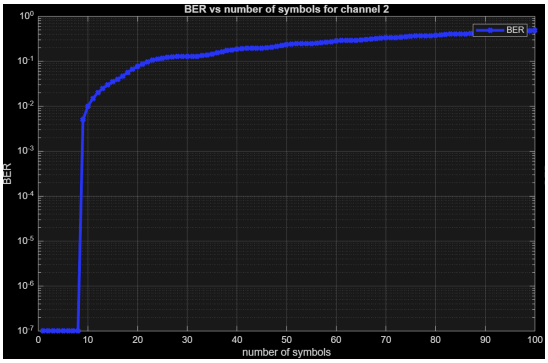
(a) BER vs SNR plot of channel 1 (50 symbols) (b) Received symbols with -1 SNR

Figure 2.1

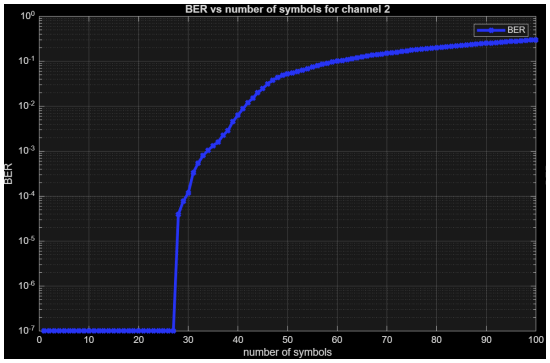
2.2 Phase tracking Performance

Figure 2.2 shows the BER as a function of the number of sent symbol with and without subcarrier phase tracking. Both case show that the BER is almost 0 for the first symbols before increasing very fast. We can also see that without additional phase correction, we can send up to 8 symbols with an almost 0 BER and that this value increases to 27 symbols with a phase tracking algorithm (Viterbi-Viterbi with filtering term $\alpha = 0.99$). In the case of channel 2, we will see in 2.3.1 that there is not much noise, but the phase is changing faster and faster. Thus a high alpha (= low filtering) can track the faster phase change better.

The tracking method fails after a certain number of OFDM symbols because the phase rotation is accelerating over time (Doppler effect), meaning that the difference in angle between two consecutive symbols gets larger and larger. In fact, since a QPSK constellation is used, the Viterbi-Viterbi algorithm assumes a phase change of less than $\pm\frac{\pi}{4}$



(a) without phase estimation, 8 symbols received without BER



(b) with phase estimate (alpha = 0.99), 27 symbols received without BER

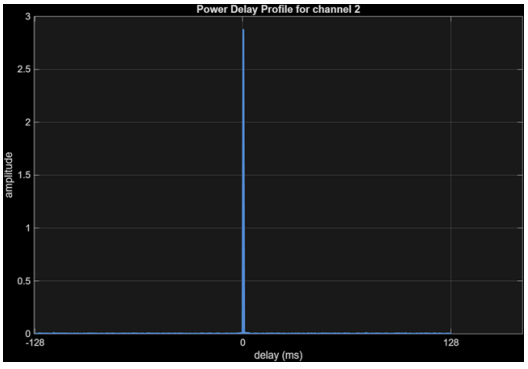
Figure 2.2: BER vs number of symbols for channel 2

between two consecutive symbols. In our case, around the 27th symbol, the channel rotates so fast that the phase shift between two symbols exceeds $\pm\frac{\pi}{4}$ leading to incorrect phase for all subsequent symbols, thus increasing the BER.

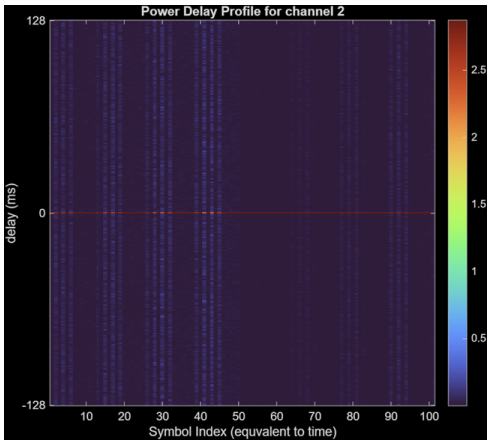
2.3 Characteristics of the emulated channels

2.3.1 Channel 2

This channel corresponds to the phase-tracking scenario. We observe that the power delay profile is ideal, exhibiting no echoes (Fig. 2.3a), and that it remains approximately constant over time (Fig. 2.3b). The interesting characteristic of this channel is the phase, which is accelerating over time. This acceleration is clearly visible in Fig. 2.4a (The phase is bounded between $-\pi$ and π). Moreover, this phase acceleration is uniform across all frequencies (Fig. 2.4b). This behavior is indicative of a Doppler effect scenario, corresponding to relative motion (or more precisely, an acceleration in this case) between the transmitter and the receiver.

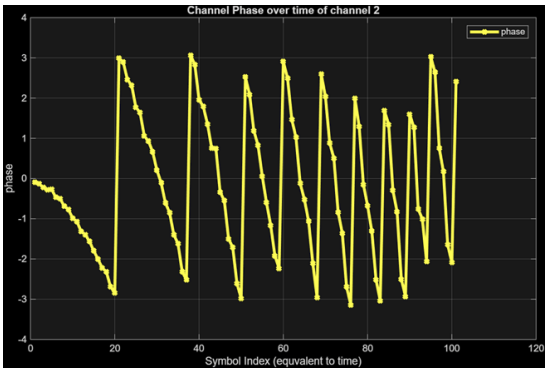


(a) channel 2 Power Delay Profile

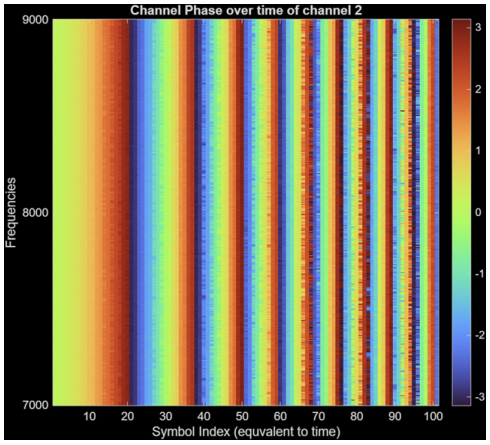


(b) PDP over time for channel 2

Figure 2.3: Channel 2 specifications (1)



(a) channel 2 phase over time



(b) Phase shift over time for channel 2

Figure 2.4: Channel 2 specifications (2)

2.3.2 Channel 3

In this channel we observe a three-peak power delay profile (Fig. 2.5), indicating the presence of two secondary echoes arriving after the main signal. The channel remains constant over time (Fig. 2.6a). No phase variation is observed in this channel (Fig. 2.6b). This scenario corresponds to an ideal multipath case in which the transmitted signal is reflected twice, producing two delayed replicas of the original signal.
A BER of 0% is acheved by our implemetation.

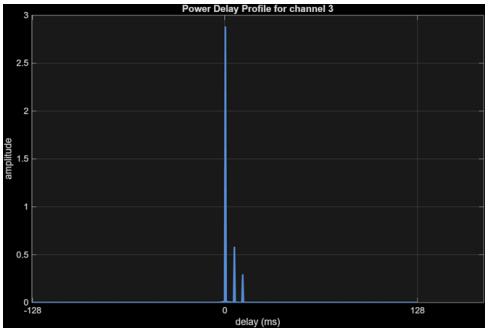
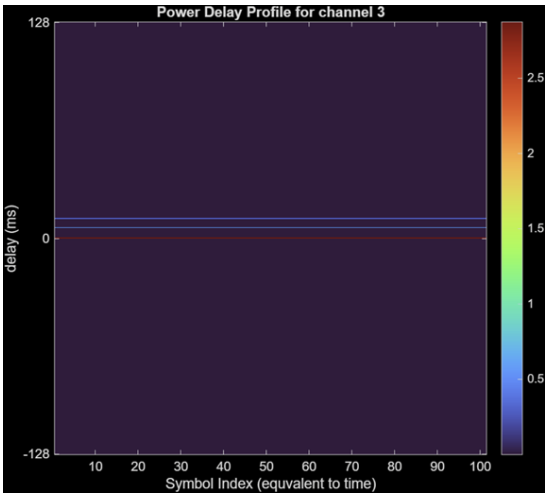
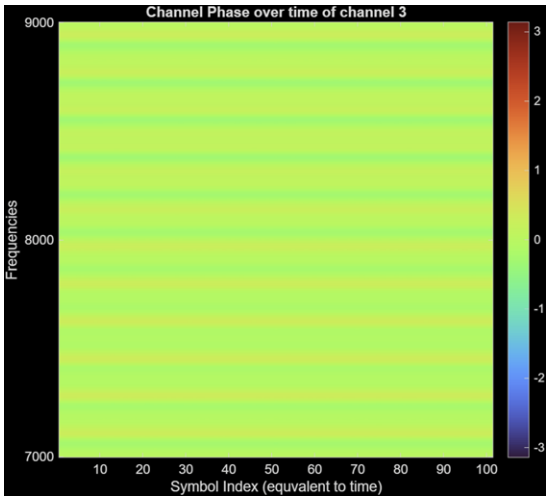


Figure 2.5: channel 3 Power Delay Profile



(a) PDP over time for the channel 3



(b) Phase shift over time for channel 3

Figure 2.6: PDP / Phase shift of channel 3

2.3.3 Channel 4

This channel was the most troublesome for us. Even with phase tracking and phase estimation, The BER would be constantly around 24%, which coresponds to half of the received bit being correct and the other half being random. Also, this channel required a lower synchronization threshold compared to the others channels.

By looking at the different plots, we can try to get a better idea of what is this channel doing.
Firstly, on the phase of the channel over time graph (Fig. 2.7), we can see that some frequencies looks completely random, but the other are constant. We can also compare the spectrum of the send and the received signal (Fig. 2.8 and 2.9). Here we again observe that some frequencies seems blocked as there are not appearing in the received signal spectrum. So the channel might act as a filter on some frequencies.

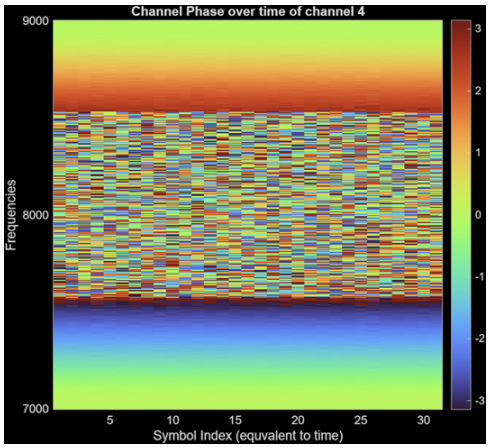


Figure 2.7: Phase shift over time for channel 4

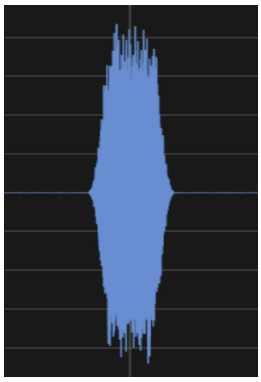


Figure 2.8: Spectrum before transmission

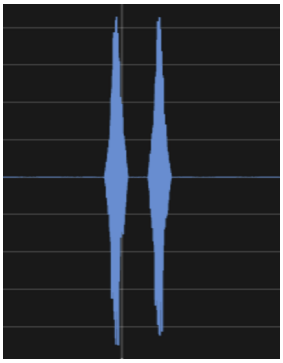


Figure 2.9: Spectrum at reception

Then, we can look at the Power delay profile (Fig. 2.10), we observe that it has a quasi symmetric shape, with some sinc-like ripples. The positive and negative delay tails extends quite a lot. We also see that it constant through time (Fig. 2.11). This sinc-like shape could be a result of the previously mentioned filtering of this channel.

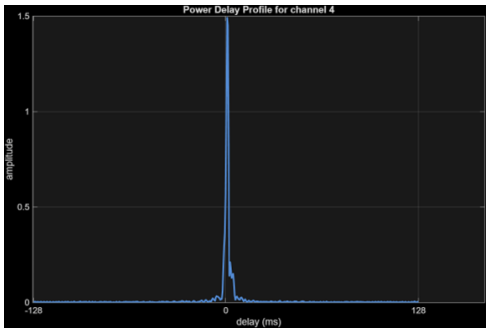


Figure 2.10: channel 4 Power Delay Profile

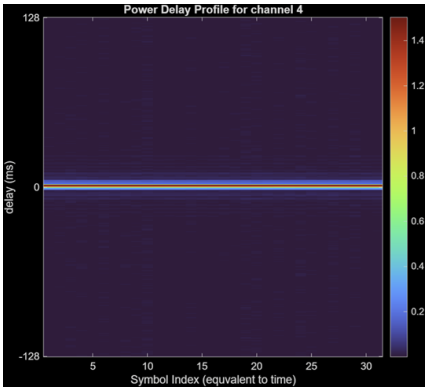


Figure 2.11: PDP over time for channel 4

To improve the BER of this channel, we tried to set the synchronization a little earlier than what the frame sync found to shift the PDP into the positive delay. This solution indeed worked really well as it gave 0% BER for channel 4 and still gave 0% to channel 3 and 5.

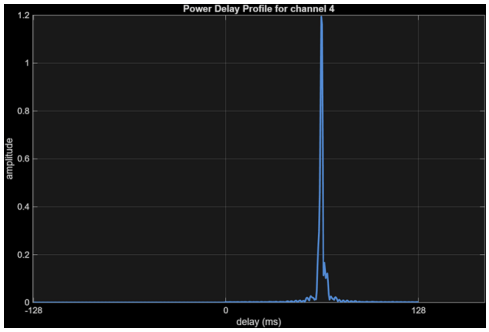


Figure 2.12: channel 4 Power Delay Profile with early sync

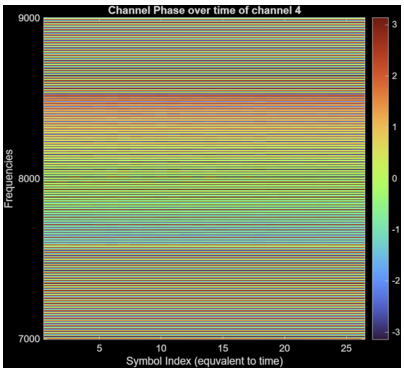
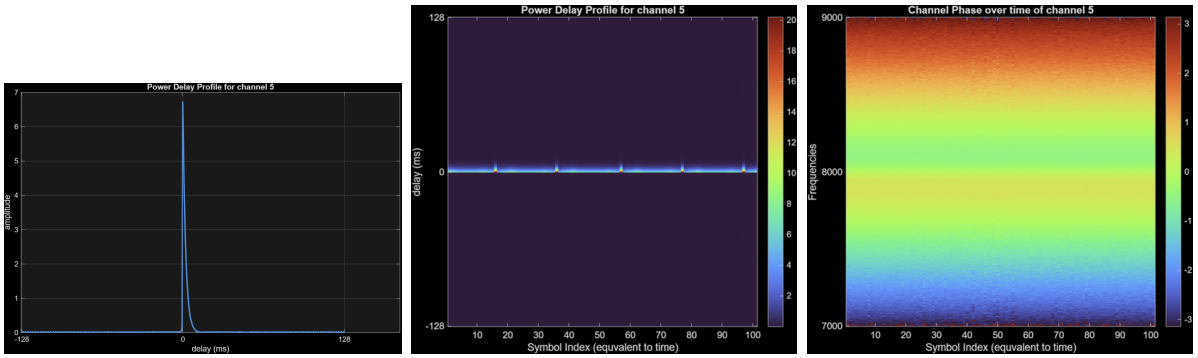


Figure 2.13: Phase shift over time for channel 4 with early sync

2.3.4 Channel 5

From the power delay profile (PDP) shown in Fig. 2.14a, we observe that Channel 5 exhibits a multipath propagation scenario. The continuously decreasing PDP (because of lots of small echos) suggests that this channel likely corresponds to an indoor transmission environment. The phase remains constant over time (Fig. 2.14c), making it easy to compensate using an inital channel estimation. However, this is not the case for the PDP. We observe that some transmitted symbols experience sudden spikes in amplitude as seen in the PDP over time (Fig. 2.14b). These spikes occur at regular time intervals, although their origin is unknown.



(a) channel 5 Power Delay Profile

(b) PDP over time for channel 5

(c) Phase shift over time for channel 5

Figure 2.14: Channel 5 specifications

In our implementation, the system is robust to these amplitude spikes because QPSK encodes information in the phase. However, if a modulation scheme that encodes information in both amplitude and phase (such as 16-QAM) were used, these amplitude fluctuations would cause performance degradation (see 2.10).

Using our implementation, we achieved a 0% bit error rate (BER) over 50 transmitted symbols.

2.4 Spectral Efficiency Analysis

1. **Calculate the spectral efficiency of your system (in bit/s/Hz).**
   The spectral efficiency (η) quantifies the information rate that can be transmitted over a given bandwidth. In other words, it measures how many bits of information can be transmitted per second for every Hertz of bandwidth occupied. It is defined as follows :
   $$\eta = \frac{R_{bit}}{B} = \frac{N_{bit,OFDM}}{T_{bit,OFDM}.B} \quad [\text{bit/s/Hz}] \tag{2.1}$$
   With R_{bit} the useful bit rate (net bit rate), B the system's bandwidth, $T_{bit,OFDM}$ the duration of one OFDM symbol and $N_{bit,OFDM}$ the number of bits transmitted per OFDM symbol.
   To compute it we use the default parameters :
   - **Bandwidth (B):** 2000 Hz
   - **Total Subcarriers (N):** 512
   - **Cyclic Prefix Length (N_{cp}):** 256 samples
   - **Modulation:** QPSK ($M = 4$, thus $\log_2(4) = 2$ bits/symbol)
   - **Active Subcarriers (N_{active}):** 512 (Assuming all subcarriers are used for data)

   Then we get :
   $$T_{bit,OFDM} = T_{useful} + T_{cp} = \frac{N}{B} + \frac{N_{cp}}{B} = \frac{512}{2000} + \frac{256}{2000} = 0.256 + 0.128 = 0.384 \text{ s} \tag{2.2}$$
   $$N_{bit,OFDM} = N_{active} \times \log_2(M) = 512 \times 2 = 1024 \text{ bits} \tag{2.3}$$
   $$R_{bit} = \frac{1024 \text{ bits}}{0.384 \text{ s}} \approx 2666.7 \text{ bit/s} \tag{2.4}$$
   Finally, the spectral efficiency is:
   $$\eta = \frac{2666.7}{2000} \approx \mathbf{1.33} \text{ bit/s/Hz} \tag{2.5}$$
2. **Which parameters are involved ?**
   The spectral efficiency depends on the following system parameters:
   (a) **Modulation Order (M):** Defines the constellation size, which determines the number of bits per subcarrier ($k = \log_2 M$). For example, QPSK ($M = 4$) yields 2 bits/complex symbol, while 16-QAM ($M = 16$) yields 4 bits.
   (b) **Ratio of Useful Time to Total Time ($N/(N + N_{cp})$):** The Cyclic Prefix represents overhead. A shorter CP relative to the symbol length increases efficiency.
   (c) **Subcarrier Utilization (N_{active}/N):** Using fewer subcarriers (e.g., zeroing edges for guard bands) reduces the efficiency.
3. **How can you improve the spectral efficiency of your system ? Discuss possible improvements and their limitations.**
   To improve the spectral efficiency of the system, we can adjust the parameters identified above. A trade-off exists for each one :

- **Increasing Modulation Order (e.g., 16-QAM or 64-QAM):** This would allow to transmit more bits per symbol, thus increasing the throughput. However, higher order constellations are more sensitive to noise. This requires a higher SNR to maintain the same BER, which can lead to transmission failure in noisy channels.
- **Reducing the Cyclic Prefix (N_{cp}):** It allows to allocate more time to data transmission. However, if the CP is shorter than the **Channel Impulse Response (CIR)**, the signal will be subject to **Inter-Symbol Interference (ISI)**, thus drastically increasing the BER.
- **Increasing the Number of Subcarriers (N) while keeping N_{cp} fixed:** This makes the CP overhead a smaller percentage of the total symbol duration, but the system becomes more sensitive to time-variations. This is because the symbols are sent on a longer period which can cause problems in dynamic acoustic channels (e.g., moving microphone) for example.

4. **Remark on the Overhead :**
   The calculated η represents the efficiency of the OFDM payload symbols. However, the effective spectral density of the entire system is lower. This is mainly due to the **preamble** and the **training Sequences**. They are both necessary to ensure a robust frame synchronization and symbol demodulation, however, they extend the transmission time without contributing to the data payload. This can become non negligible, especially in the case of small number of sent symbol per frame.

2.5 Acoustic channel

Our system in a real acoustic channel
In a quiet environment, BER of 0% are possible. Fig. 2.15 shows PDP in a quiet room. We have also observed that distance is usually not a problem as long as synchronization is possible (threshold is not too high). Longer bit sequences require phase tracking as the phase may shift during long transmission. We have verified that moving the microphone around makes the phase rotate, as previously mentioned in channel 2. We also tried to recreate the channel 2 characteristics by accelerating the microphone away from the speakers. We obtain a similar phase plot (Fig. 2.16), thus verifying our assumption on channel 2.

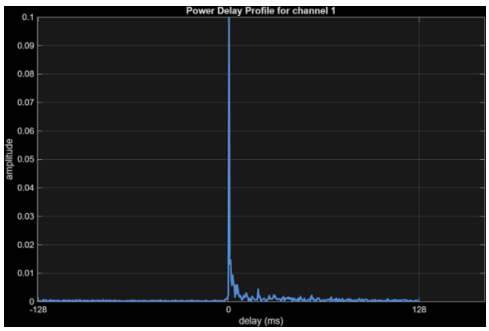


Figure 2.15: PDP in a quiet environment

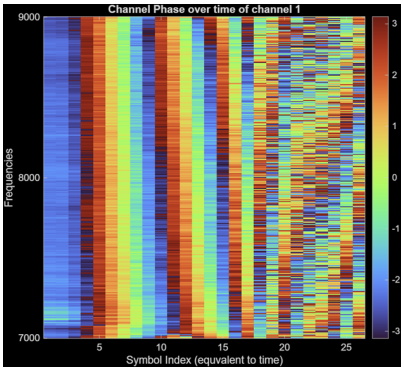
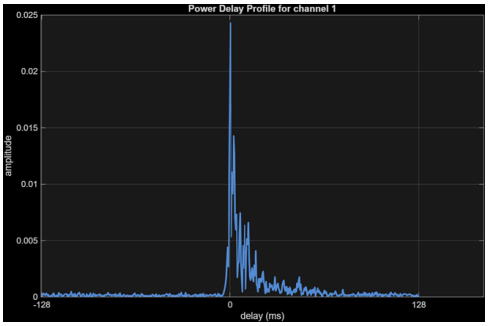


Figure 2.16: Phase graph of the recreation of channel 2

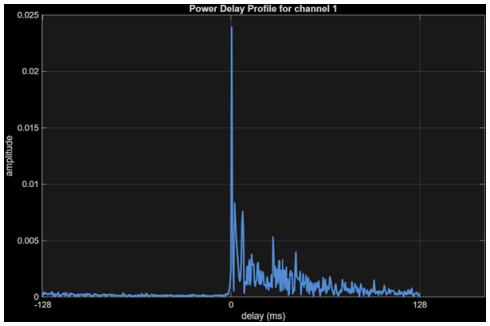
Another thing we did was transmission between two PCs, one with the speaker and one

with the microphone, and it worked successfully. Then we also tried more difficult environments: we put the microphone under a jacket to attenuate the sound, which did not cause any drop in reception accuracy, and we also tried speaking next to it during the transmission. Again, we observed almost every time a perfect BER. This is probably because our voice is far from the 8 kHz range, so it does not interfere with the transmission.

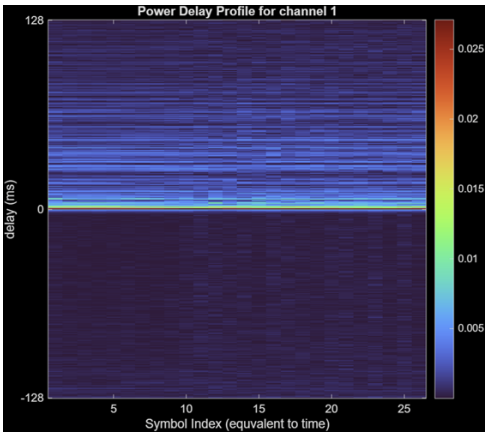
Finally, we tried more extreme environments: Fig. 2.17a shows a power delay profile under a table to accentuate reflections and echoes. We also tried transmitting with obstacles, ambiant sounds and distance, which produced the plots 2.17b to 2.17d. Those latter examples produced varying results from close to 0 BER to around 30% BER (counting only the runs were the synchronization was achieved).



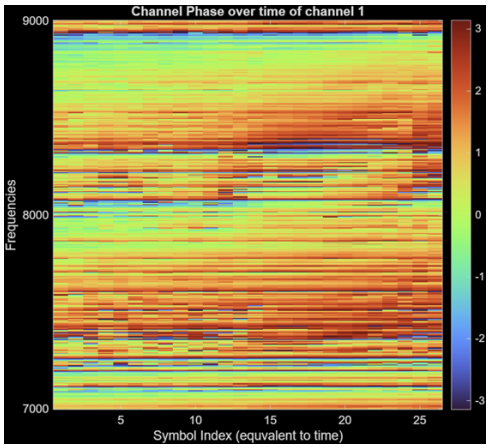
(a) PDP of transmission under a table



(b) PDP of harsh transmission



(c) PDP over time of harsh transmission



(d) phase over time of harsh transmission

Figure 2.17

2.6 Text transmission

We also implemented a message transmission system where the user is asked to type an input text. The text string is encoded in ASCII and then with the same LFSR encoding as the image (See section 2.7) to reduce the peak-to-average power ratio. The text transmission can be tested in the *audiotrans_comm_protocol.m* file by putting the **conf.id** parameter to the value of **2**.

2.7 Image transmission

We also implemented image transmission. We preprocessed our images to be 128x128 pixels on 4bit greyscale to reduce the numbers of bits needed. The transmission of an image takes 64 OFDM symbols, that takes $0.384 \times 64 = 24.576$ sec of transmission.

However, one issue needs to be tackled : OFDM has a really high peak-to-average power ratio if the image bits are send as they are, because of the constructive interferences between similar bits will occur. Our solution was to encode the image using the same LFSR sequence used for the preamble, The sequence is XORed with the image bit, and the same operation is repeated at the Rx to decode the image (2.19). The effect of this encoding can be observed in the spectrum (2.20 and 2.21), which shows high peaks without encoding, and a flatter spectrum with the encoding. The image transmission can be tested in the *audiotrans_image.m* file or in the *audiotrans_comm_protocol.m* file by putting the **conf.id** parameter to the value of **1**.

Figure 2.18: Image sent on the channel 2 (64 symbols)

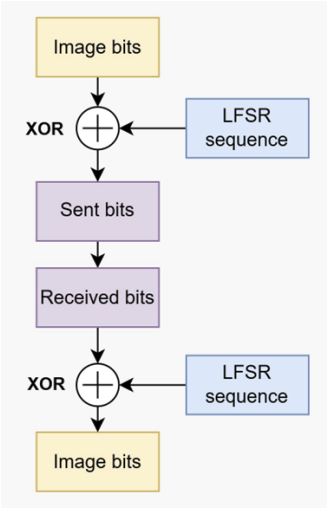


Figure 2.19: LFSR encoding / decoding diagram

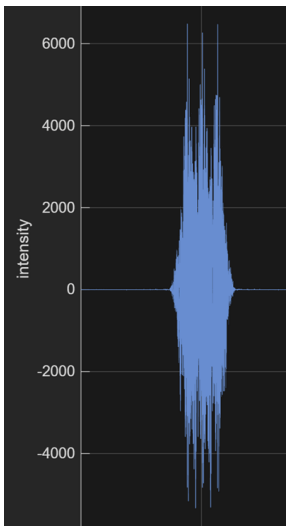


Figure 2.20: Spectrum without encoding

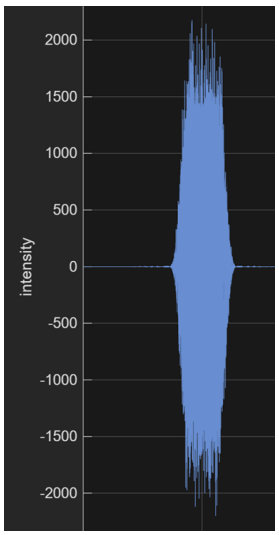


Figure 2.21: Spectrum with encoding

2.8 Communication Protocol

A communication protocol has been implemented by slightly modifying the code and the data architecture (Figure 2.22) illustrates this new structure. A block composed of two OFDM symbols was inserted between the preamble and the payload data, consisting of a training sequence followed by a header. The training sequence is used to equalize the header symbol, ensuring its robust transmission with minimal error. The header symbol (see Figure 2.23) contains the following fields:

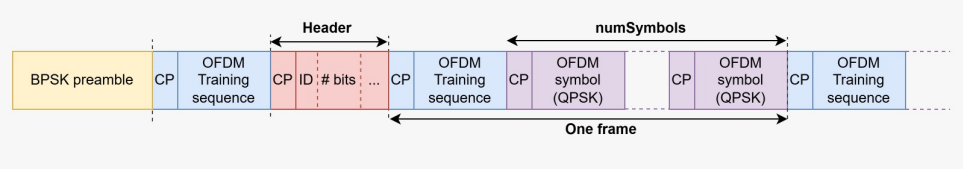


Figure 2.22: New data architecture

- **ID (2 bits):** These bits define the type of data being transmitted. The ID encoding is defined as follows:
  1. **ID = 0 = 00b**: Raw bitstream.
  2. **ID = 1 = 01b**: Image transmission. (See Section 2.7 for more details)
  3. **ID = 2 = 10b**: Text transmission. (See Section 2.6 for more details).
- **Data Length (32 bits):** Encodes the total number of useful payload bits (that are sent). A 32-bit field allows indexing up to 4.29 billion bits ($2^{32}-1 \approx 4.29 \times 10^9$), which is sufficient to cover all supported data types (images, text, or raw streams).
- **Cyclic Redundancy Check (CRC - 8 bits):** A checksum calculated over the preceding 34 bits (ID + Length). The receiver performs the same mathematical operation on the received header. If the calculated result matches the received CRC bits, the header is validated, allowing the receiver to safely extract the payload using the decoded ID and length. If it doesn't, an error message is displayed and the code is stopped.
- **The rest :** The remaining bits are set to random values to avoid clipping. If needed, they can be used to add other information such as the number of symbols per frame, the modulation order, ...

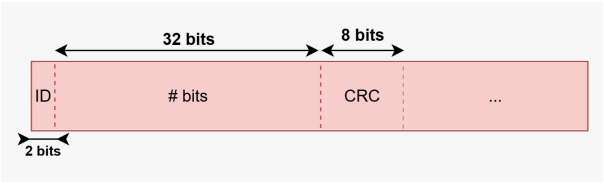


Figure 2.23: Header symbol architecture (without CP)

This protocol allows the system to transmit various data types dynamically, without requiring the receiver to have prior knowledge of the transmission content or duration. To test this protocol, run the ***audiotrans_comm_protocol.m*** matlab code (that uses the ***rxofdm_comm_protocol.m*** and ***txofdm_comm_protocol.m*** codes)

2.9 Multiple channel estimation

We experimented with performing channel estimation multiple times throughout the transmission, by sending multiple frames each one composed of a training sequence and multiple OFDM data symbols, to improve performance. However, since the channel remained mostly constant in the emulated and audio channels, this approach did not lead to a significant improvement. Nevertheless, we expect that in a real-world scenario it would be beneficial. For example, if a user moves into a building, the channel characteristics would vary over time, and such an implementation could track these changes and adapt accordingly. This approach is implemented in the *audiotrans_comm_protocol.m* file.

2.10 Throughput optimization

In order to maximize throughput, we can use a higher modulation order scheme using M-QAM. In 16-QAM, 4 bits are encoded in each constellation points, that is twice more than in a QPSK mapping. Using 64-QAM (8 bits per symbol) we acheived a BER of 6.8% on channel 5. This higher BER is due to the change in amplitude of the channel (pulsing effect we saw in 2.3.4) since information is also encoded in the amplitude with M-QAM. We find that :

$$\text{Throughput} = \frac{\text{number of bits}}{\text{time}} = \frac{8 \text{ bits} \times 512}{0.384 \text{ s}} = 10667 \text{ bits/sec} \tag{2.6}$$

which is 4 times the original throughput using QPSK.

We could also reduce the cyclic prefix length to optimize even more the throughput, although it has less of an impact as changing the modulation order. Using 64-QAM and a cyclic prefix length of 201 gives us a BER of 9.6% on channel 5. This corresponds to

$$\text{Throughput} = \frac{\text{number of bits}}{\text{time}} = \frac{8 \text{ bits} \times 512}{\frac{512+201}{2000}} = 11489 \text{ bits/sec} \tag{2.7}$$

M-QAM has been implemented in the *audiotrans_MQAM.m* file.

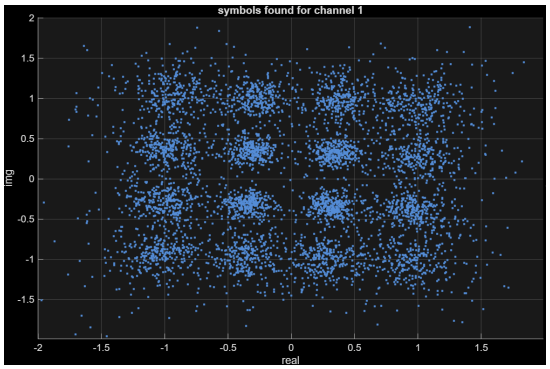


Figure 2.24: 16-QAM in audio channel

2.11 Possible future improvement

2.11.1 Different Channel Estimation methods

The current channel estimation method uses a **Block-Type Pilot Arrangement** (see section 1.4.1). This arrangement is particularly efficient for channels with slowly varying phase and amplitude (frequency-selective fading).
Alternative channel estimation methods exist for different channel conditions, including:

- **Comb-Type Pilot Arrangement (Horizontal):** Pilot tones are inserted at specific subcarriers for every symbol period. This allows the receiver to track fast time-varying channels (high Doppler spread), but requires interpolation in the frequency domain to estimate the channel for data subcarriers.
- **Scattered Pilots:** Pilots are distributed in a diamond or lattice pattern across the time-frequency grid. This method provides a balance, capturing both frequency selectivity and time variation (doubly-selective channels). However, it requires complex 2D interpolation.

- **Decision-Directed Channel Estimation (DDCE):** This method uses the re-modulated decisions of the previous data symbols to update the channel estimate. It maximizes throughput but can suffer from error propagation if the initial decisions are incorrect.

2.11.2 Different Phase Tracking methods

The current phase tracking algorithm is the **Viterbi-Viterbi** algorithm (see section 1.4.2). This is a blind (non-data-aided) estimation method that works by raising the received signal to the M-th power (where M is the modulation order) to eliminate the data modulation. While this method is computationally efficient, it suffers from noise enhancement at low SNR and struggles to track the phase in fast time-varying channels. Further experiments could include the use of the following algorithms:

- **"Horizontal channel estimate" :** A simple idea for channel 2 (but uniquely for channel 2, it does not really have other use cases) is to use one of the 512 subcarriers as a known symbol to estimate the phase. By knowing what the constellation point should be, we can easily calculate the phase shift of the channel and then applying it to the other subcarriers. This only works because the phase change is homogeneous throughout the frequencies in channel 2. We choose not to implement it as it is not really useful for other uses.
- **Decision-Directed Phase Locked Loop (DD-PLL):** A feedback-based method that uses the demodulated decisions from previous symbols to estimate the phase error of the current symbol. This allows for smooth tracking of time-varying phase drift but is susceptible to error propagation if incorrect decisions are fed back into the loop.

2.11.3 MMSE Equalizer (vs. Zero-Forcing)

The current system employs **a Zero-Forcing (ZF) equalizer** $(1/H)$, which is simple to implement but suffers from a key limitation: it amplifies noise at frequencies where the channel response is weak (e.g., deep fades). A more robust alternative would be a **Minimum Mean Square Error (MMSE) equalizer**, which, unlike ZF, balances interference suppression with noise minimization. However, MMSE is computationally more complex, as it requires real-time estimation of the noise power (σ^2_{noise}) and Signal-to-Noise Ratio (SNR), typically via a Noise Variance Estimator.

2.11.4 Forward Error Correction (FEC)

Currently, the system uses uncoded modulation, meaning that if a bit is corrupted by the channel, it is permanently lost. To improve reliability, **Forward Error Correction (FEC)** could be added (e.g., Convolutional Codes or Hamming Codes). This technique involves adding redundant bits to the transmitted data stream. These extra bits allow the receiver to mathematically detect and correct a certain number of errors without asking for re-transmission. While this reduces the effective data throughput, it significantly reduces the Bit Error Rate (BER) in noisy conditions.