Cédric Colin – Marvyn Levin

DosOok | Comparison Report of Low-Pass Filters

TEAM CHARTREUSE

1 2 7 2 5 5 0

Nord Franche-Comté Belfort - Montbéliard
UNIVERSITÉ DE FRANCHE-COMTÉ

2023 - 2024 | Semestre 1

Table of contents

Introduction

The study aims to design and implement two programs, DosSend and DosRead, enabling the exchange of textual data through an audio channel. This innovative approach utilizes sound waves, providing an alternative to conventional technologies. The article meticulously details the employed methods, presents the obtained results, and offers a comprehensive discussion on the relevance and prospects for improving this technology.

Firstly, we will explore the section describing the use of Amplitude Shift Keying (ASK) in audio transmission. Secondly, we will present the initial hypothesis by showcasing the data obtained through ASK transmission. Finally, we will analyze the results in detail.User interface.

1. User Interface

Before diving into the underlying algorithms, let's quickly review the user interface parameters to facilitate efficient use of the DosOok program. The program is configured with a graphical user interface, offering a comprehensive user experience. The program execution can be done using the command 'java DosSend' for encoding or 'java DosRead ./file.wav' for decoding.

2. Description of Algorithms

We detail two filters, LPFilter1 and LPFilter2, used in DosOok to optimize audio transmission. Here is an audio in figure 1 on which we have applied low-pass filters in figure 2.

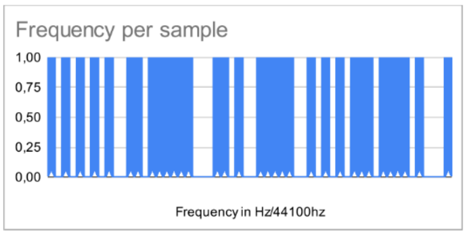


Figure 1: Audio in chinese

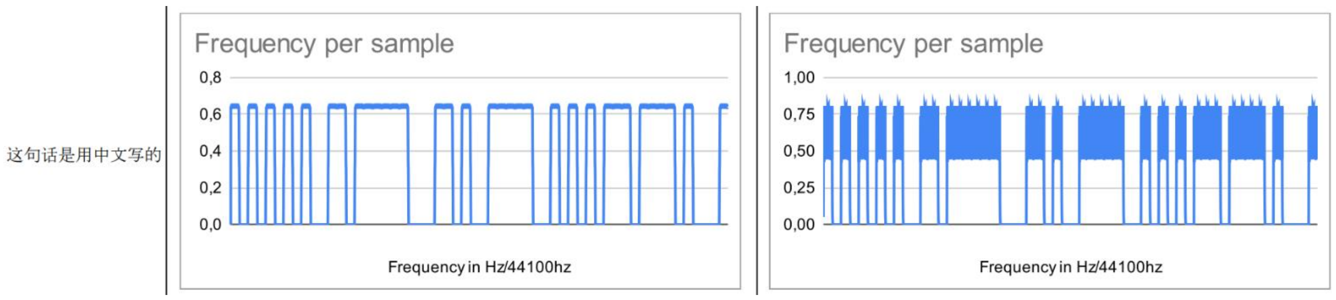
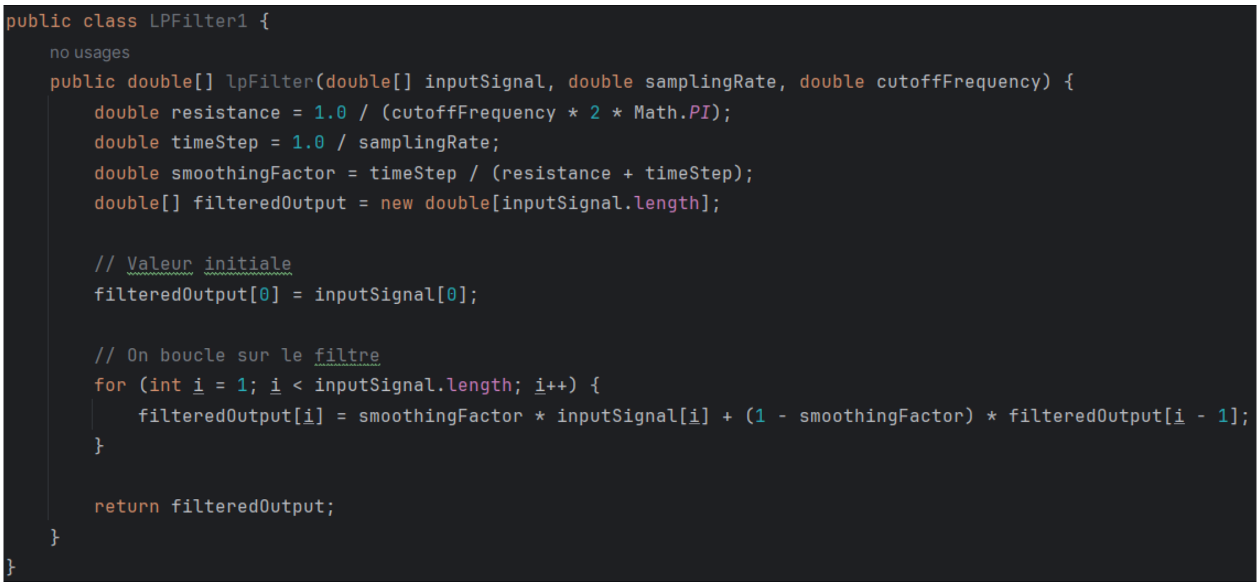


Figure 2: Audio after passes LPFilters

2.1 LPFilter1

A first-order filter, LPFilter1, operates with a classical low-pass filtering approach, the Moving Average. It is designed to attenuate undesirable high frequencies, improving the overall quality of sound transmission. Its simplicity makes it effective for basic situations.

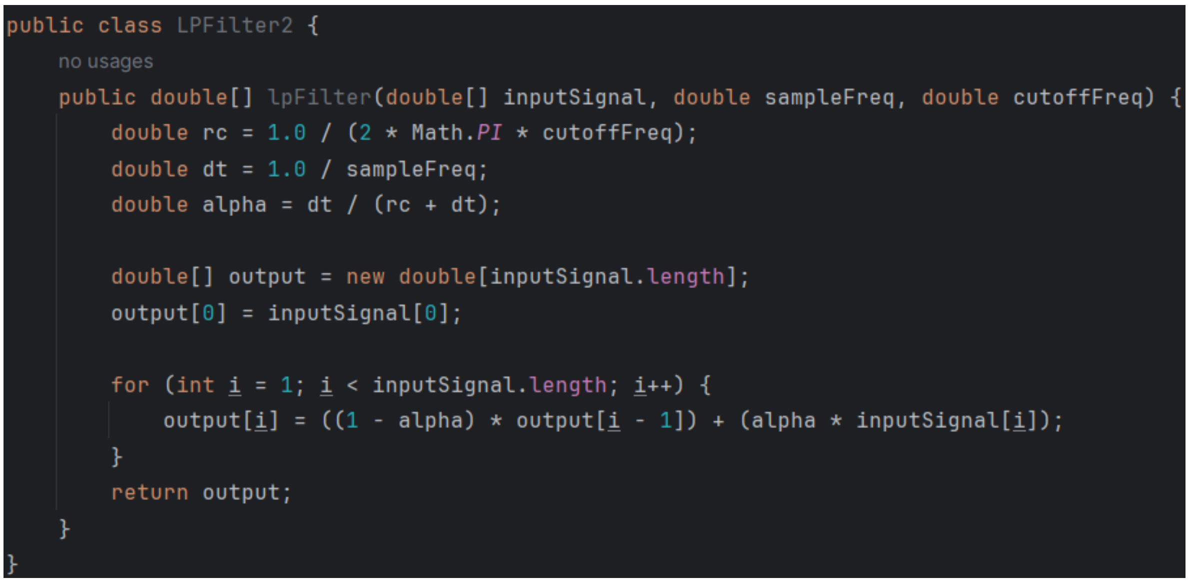


```java
public class LPFilter1 {
    no usages
    public double[] lpFilter(double[] inputSignal, double samplingRate, double cutoffFrequency) {
        double resistance = 1.0 / (cutoffFrequency * 2 * Math.PI);
        double timeStep = 1.0 / samplingRate;
        double smoothingFactor = timeStep / (resistance + timeStep);
        double[] filteredOutput = new double[inputSignal.length];

        // Valeur initiale
        filteredOutput[0] = inputSignal[0];

        // On boucle sur le filtre
        for (int i = 1; i < inputSignal.length; i++) {
            filteredOutput[i] = smoothingFactor * inputSignal[i] + (1 - smoothingFactor) * filteredOutput[i - 1];
        }

        return filteredOutput;
    }
}
```

LPFilter 1: Attenuation of high frequencies for basic audio transmission

2.2 LPFilter2

LPFilter2 represents an improvement over its predecessor. Introducing additional components, it offers a more accurate frequency response and better suppression of unwanted frequencies. Its sophisticated algorithm ensures optimal audio transmission, especially in more complex conditions. These filters, like the algorithms mentioned earlier, align with DosOok's overall strategy for efficient and high-quality sound data transmission.



```java
public class LPFilter2 {
    no usages
    public double[] lpFilter(double[] inputSignal, double sampleFreq, double cutoffFreq) {
        double rc = 1.0 / (2 * Math.PI * cutoffFreq);
        double dt = 1.0 / sampleFreq;
        double alpha = dt / (rc + dt);

        double[] output = new double[inputSignal.length];
        output[0] = inputSignal[0];

        for (int i = 1; i < inputSignal.length; i++) {
            output[i] = ((1 - alpha) * output[i - 1]) + (alpha * inputSignal[i]);
        }
        return output;
    }
}
```

LPFilter 2: Advanced optimization of frequency response for improved audio quality

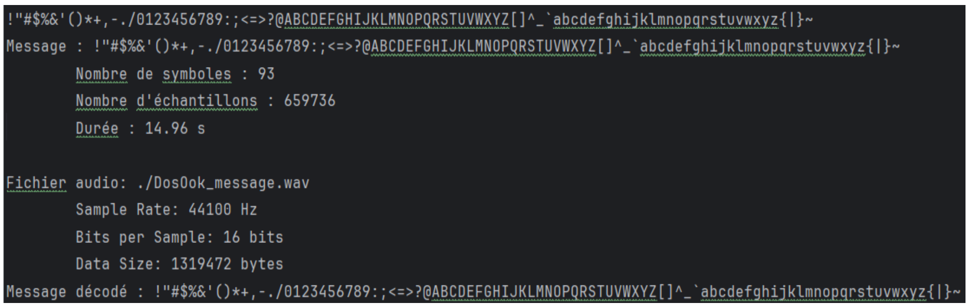
These filters, like the algorithms mentioned earlier, align with DosOok's overall strategy for efficient and high-quality sound data transmission.

3. Algorithms Comparison

3.1 Methodology

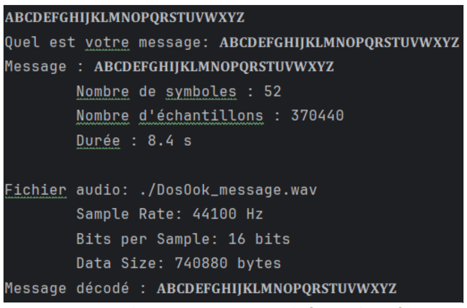
To evaluate the efficiency of audio modulation filters, LPFilter1 and LPFilter2, comprehensive tests were conducted. Each test involves the transmission of standardized textual data via DosOok. The filters are successively integrated, and each configuration is evaluated in terms of processing time.

The tests were conducted on a computer equipped with an AMD RYZEN processor, using a Linux Ubuntu environment. All ASCII characters on 8 bits (Figure 4) were tested, transitioning to Unicode characters on 16 bits (Figure 3), and tests were performed with characters on 32 bits (Figure 5) to ensure representative results.



```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz{|}~
Message : !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz{|}~
        Nombre de symboles : 93
        Nombre d'échantillons : 659736
        Durée : 14.96 s

Fichier audio: ./DosOok_message.wav
        Sample Rate: 44100 Hz
        Bits per Sample: 16 bits
        Data Size: 1319472 bytes
Message décodé : !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

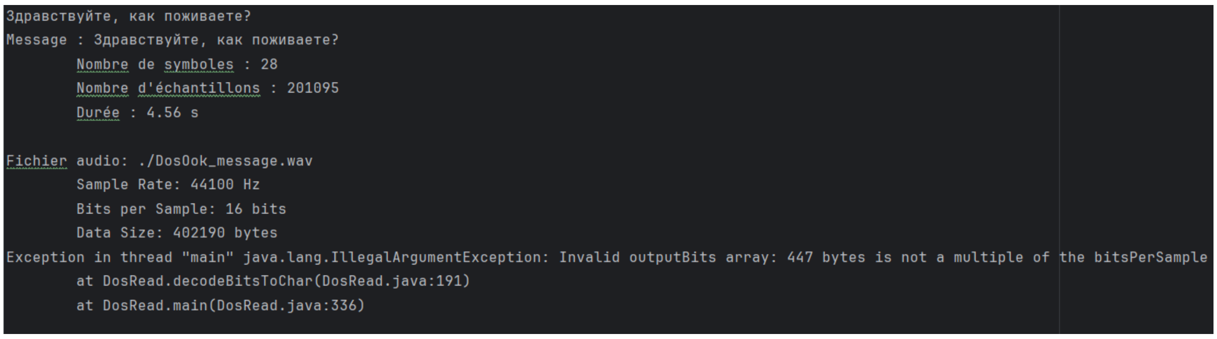
Figure 4: Test with all ASCII characters



```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Quel est votre message: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Message : ABCDEFGHIJKLMNOPQRSTUVWXYZ
        Nombre de symboles : 52
        Nombre d'échantillons : 370440
        Durée : 8.4 s

Fichier audio: ./DosOok_message.wav
        Sample Rate: 44100 Hz
        Bits per Sample: 16 bits
        Data Size: 740880 bytes
Message décodé : ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Figure 3: Test with Unicode 16 bytes

However, some tests turned out to be inaccurate as we were unable to read the Russian phrase, as depicted in figure 5.



```
Здравствуйте, как поживаете?
Message : Здравствуйте, как поживаете?
        Nombre de symboles : 28
        Nombre d'échantillons : 201095
        Durée : 4.56 s

Fichier audio: ./DosOok_message.wav
        Sample Rate: 44100 Hz
        Bits per Sample: 16 bits
        Data Size: 402190 bytes
Exception in thread "main" java.lang.IllegalArgumentException: Invalid outputBits array: 447 bytes is not a multiple of the bitsPerSample
        at DosRead.decodeBitsToChar(DosRead.java:191)
        at DosRead.main(DosRead.java:336)
```

Figure 5: Test with Unicode 32 bytes

3.2 Results

The in-depth evaluation of the performance of LPFilter1 and LPFilter2 reveals substantial differences, particularly in terms of execution time. The ratio between the execution time of LPFilter1 and LPFilter2 varies significantly in each trial, indicating notable disparities in the efficiency of these filters in figure 6.

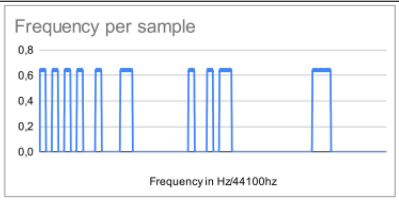
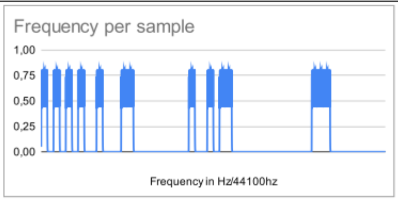
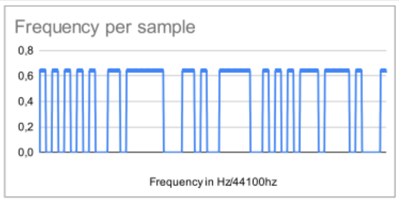
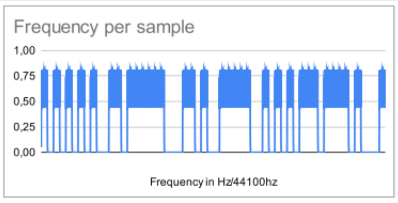
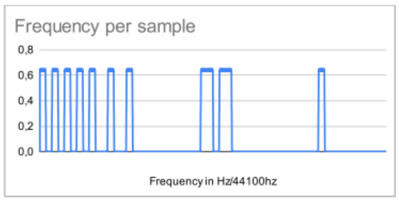
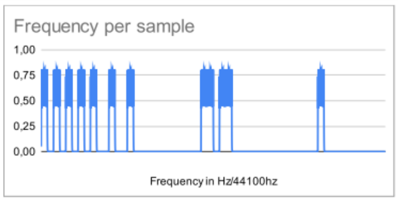
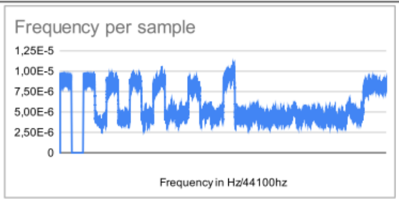
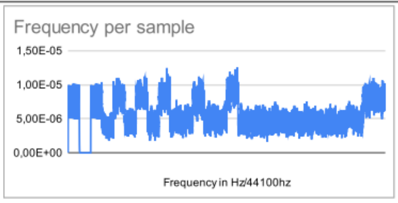
/	Results				
Message	Filter n°1 Results	Filter n°2 Results	Time for filter n°1(ms)	Time for filter n°2 (ms)	Time 1/Time 2 ratio [1]
Bip			2,777	0,5783	4,802005879
这句话是用中文写的			4,661	1,814	2,569459757
Il neige			4,389	1,525	2,878032787
Il neige (version avec audio brouillé)			7,8	1,518	5,138339921

Figure 6: Sheet with time of execution for LPFilters

These ratios demonstrate that LPFilter2 is significantly faster than LPFilter1 in each evaluation scenario. Higher ratio values indicate a clear improvement in the performance of LPFilter2 compared to LPFilter1. This disparity can be attributed to intrinsic characteristics of the filters and their impact on audio modulation.

It is essential to emphasize that the speed of LPFilter2 does not compromise the transmission success rate. While the processing speed is increased, the preservation of data integrity is maintained, giving LPFilter2 a notable advantage in optimizing DosOok.

Conclusion

In conclusion, there exists a substantial performance gap between the two filters, with the second algorithm proving to be the fastest. Specifically, it is approximately 3.8475 times faster, based on the following collected ratios:

- Ratio 1: 4.80,
- Ratio 2: 2.57,
- Ratio 3: 2.88,
- Ratio 4: 5.14.

It is also noteworthy that reliability remains consistent in both algorithms.

However, it is crucial to acknowledge limitations observed in the decoding process (DosRead). Challenges arise in handling characters such as Russian and accents, potentially leading to data loss. Additionally, an unforeseen constraint emerges, as characters encoded on 32 bits cannot be efficiently processed during both encoding (DosSend) and decoding. Despite these challenges, DosOok represents a significant advancement in sound-based data transmission, setting the stage for further refinement and future developments in the field.

List of Illustrations