Indian Institute of Science Education and Research Bhopal

Quadrature Amplitude Modulation (QAM)

ECS301 - Principles of Communication

Instructor: Dr. Ankur Raina

Sattwik Sahu

Roll No. 21241

sattwik21@iiserb.ac.in

April 17, 2024

Contents

Chapter 1

Introduction

Overview of Quadrature Amplitude Modulation

Quadrature Amplitude Modulation (QAM) is a modulation technique that combines two amplitude-modulated (AM) signals into a single channel, thereby increasing the efficiency of bandwidth utilization. Unlike its application in digital systems, QAM in analog contexts involves the modulation of analog signals. This is achieved by varying the amplitude of two carrier waves, typically sinusoids, that are 90 degrees out of phase. These are referred to as the in-phase (I) and quadrature (Q) components.

Historical Background

Originally developed for use in radio communication systems, the concept of QAM has been around since the early 20th century. It was first used in analog television broadcasting and later found applications in professional video transmission systems, where it helped to maximize the amount of video information transmitted over limited bandwidth channels.

Importance in Modern Analog Communication Systems

In the realm of analog communication, QAM is particularly significant because of its ability to carry twice the bandwidth compared to a standard AM signal. This is crucial in systems where bandwidth is limited and must be utilized as efficiently as possible.

Application in Analog Television

One of the most notable applications of analog QAM is in the transmission of analog color television signals. Here, QAM allows for the transmission of two primary color signals simultaneously, thereby efficiently using the available bandwidth. In the NTSC and PAL color TV systems, the chrominance (color information) is transmitted using a QAM subcarrier on top of the main video carrier.

Application in Radio and Satellite Transmissions

QAM is also used in some analog radio communication systems and satellite transmissions, where it helps in the efficient use of bandwidth by allowing multiple analog signals to be transmitted simultaneously on the same carrier frequency. This technique is beneficial for long-distance communication where bandwidth is at a premium.

Challenges in Analog Systems

Implementing QAM in analog systems poses several challenges, primarily related to signal integrity and noise. Since QAM signals are more susceptible to noise and interference compared to simpler modulation schemes, maintaining high-quality transmission requires careful design and often more sophisticated equipment. The alignment of phase and amplitude in the in-phase and quadrature components is critical, and any mismatch can lead to significant degradation in signal quality.

Chapter 2

Theory

Introduction

Quadrature Amplitude Modulation (QAM) [1] is a form of modulation used in both analog and digital communication systems. In analog systems, QAM combines two amplitude-modulated signals into a single channel, thereby increasing the efficiency of data transmission. This document explores the foundational theory behind analog QAM.

2.1 Basic Principles of QAM

QAM involves the modulation of a signal with two carriers that are out of phase by 90 degrees. These carriers are known as the in-phase (I) and the quadrature (Q) components.

2.1.1 Signal Representation

The general expression for a QAM signal can be written as follows:

$$s(t) = I(t)\cos(\omega_c t) - Q(t)\sin(\omega_c t) \tag{2.1}$$

where:

1. $I(t)$ is the in-phase component,
2. $Q(t)$ is the quadrature component,
3. ω_c is the angular frequency of the carrier,
4. t is time.

2.1.2 Carrier Signals

The carriers for the in-phase and quadrature components are:

$$c_I(t) = \cos(\omega_c t), \quad c_Q(t) = \sin(\omega_c t)$$

These carriers are orthogonal to each other, which is a key property that allows the independent modulation of $I(t)$ and $Q(t)$.

2.2 Modulation Process

The modulation process in QAM can be viewed as a projection of the message signals onto the two carrier waves.

2.2.1 In-Phase and Quadrature Components

The in-phase and quadrature components are typically derived from the original message signal. For a simple example, consider a message signal $m(t)$ that is split into two separate signals:

$$I(t) = m_1(t), \quad Q(t) = m_2(t)$$

where $m_1(t)$ and $m_2(t)$ can be different aspects of the data (e.g., different data streams).

2.2.2 Generation of the QAM Signal

The QAM signal is generated by combining the modulated in-phase and quadrature components:

$$s(t) = m_1(t)\cos(\omega_c t) - m_2(t)\sin(\omega_c t) \tag{2.2}$$

This equation shows how the data carried in $m_1(t)$ and $m_2(t)$ modulate the cosine and sine carrier waves, respectively.

2.3 Demodulation of QAM Signals

To recover the original message components from the QAM signal, the receiver must perform demodulation.

2.3.1 Demodulation Technique

The demodulation can be performed by multiplying the received signal by the carrier signals and then applying a low-pass filter to extract the message components:

$$I'(t) = s(t)\cos(\omega_c t) \tag{2.3}$$

$$Q'(t) = s(t)\sin(\omega_c t) \tag{2.4}$$

Low-pass filtering these products removes the high-frequency terms, leaving only $m_1(t)$ and $m_2(t)$.

Chapter 3

Implementation

3.1 Python Code

A web-based frontend was created using Streamlit which allowed uploading of 2 audio files and demonstrates the process of QAM on these files' audio signals.

The part of the code which performs modulation and demodulation of the two audio signals is given below

3.1.1 Modulation

```python
def qam_modulation(
    signal1: Signals, signal2: Signals, carrier_freq: CarrierFrequency
) -> list[float]:
    """
    Perform quadrature amplitude modulation (QAM) on two signals.

    Args:
        signal1 (Signals): The first input signal (either an AudioFile instance or a list of floats)
        signal2 (Signals): The second input signal (either an AudioFile instance or a list of floats)
        carrier_freq (CarrierFrequency): The carrier frequency and its unit.

    Returns:
        list[float]: The modulated signal.
    """
    # Convert the carrier frequency to the appropriate unit
    if carrier_freq.unit == "khz":
        freq = carrier_freq.value * 1e3
    elif carrier_freq.unit == "mhz":
        freq = carrier_freq.value * 1e6

    # Extract the signal data
    if isinstance(signal1, list):
        signal1_data = signal1
    else:
        signal1_data = signal1.data

    if isinstance(signal2, list):
        signal2_data = signal2
    else:
        signal2_data = signal2.data
```

```
31
32      # Calculate the time vector
33      t = np.arange(len(signal1_data)) / len(signal1_data)
34
35      # Generate the carrier signals
36      carrier_signal_1 = np.cos(2 * np.pi * freq * t, dtype=np.float32)
37      carrier_signal_2 = -1j * np.sin(2 * np.pi * freq * t, dtype=np.float32)
38
39      # Perform QAM modulation
40      modulated_signal_1 = np.array(signal1_data, dtype=np.float32) * carrier_signal_1
41      modulated_signal_2 = np.array(signal2_data, dtype=np.float32) * carrier_signal_2
42
43      modulated_signal = modulated_signal_1 + modulated_signal_2
44
45      return modulated_signal
```

3.1.2 Demodulation

```
1   def demodulate_signal(
2       modulated_signal: list[float], carrier_freq: CarrierFrequency, sampling_rate: int
3   ) -> Tuple[list[float], list[float]]:
4       """
5       Demodulate the modulated signal.
6
7       Args:
8           modulated_signal (list[float]): The modulated signal.
9           carrier_freq (CarrierFrequency): The carrier frequency and its unit.
10          sampling_rate (int): The sampling rate of the modulated signal.
11
12      Returns:
13          Tuple[list[float], list[float]]: The demodulated signals.
14      """
15      # Convert the carrier frequency to the appropriate unit
16      if carrier_freq.unit == "khz":
17          freq = carrier_freq.value * 1e3
18      elif carrier_freq.unit == "mhz":
19          freq = carrier_freq.value * 1e6
20
21      # Calculate the time vector
22      t = np.arange(len(modulated_signal)) / sampling_rate
23
24      # Generate the carrier signals
25      carrier_signal_1 = np.cos(2 * np.pi * freq * t, dtype=np.float32)
26      carrier_signal_2 = -1j * np.sin(2 * np.pi * freq * t, dtype=np.float32)
27
28      # Demodulate the signal
29      demodulated_signal1 = np.array(modulated_signal) * carrier_signal_1
30      demodulated_signal2 = np.array(modulated_signal) * carrier_signal_2
31
32      b, a = butter(1, 2 * carrier_freq.value / sampling_rate, btype="low", analog=False)
33      demodulated_signal1 = lfilter(b, a, demodulated_signal1)
34      demodulated_signal2 = lfilter(b, a, demodulated_signal2)
35
36      return np.real(demodulated_signal1).tolist(), np.real(demodulated_signal2).tolist()
```

3.2 Explanation

3.2.1 Modulation

1. Lines 36 and 37 create the two carrier signals with a phase difference.
2. Lines 40, 41 and 43 modulate these carriers with different message signals, as seen in Equation 2.2

3.2.2 Demodulation

1. Lines 25 and 26 generate the two local oscillator signals
2. Lines 29 and 30 demodulate the modulated signal to retrieve the two message signals, as seen in Equation 2.3 and Equation 2.4
3. A low pass filter is applied to the demodulated signals to minimize distortion.

3.3 Usage

3.3.1 Prerequisites

Python should be installed on your **Linux** machine.

3.3.2 Run the Project

To run the project, execute the following commands in your terminal (in Linux):

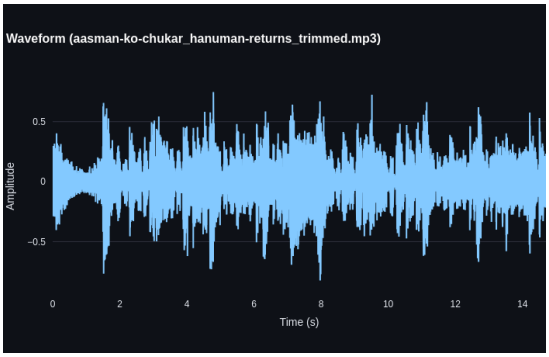
```
cd qamod
python -m venv .venv
source .venv/bin/activate
python -m pip install poetry
poetry shell
poetry install
poetry run python -m streamlit run src/qamod/__main__.py
```

This opens a browser window where you can upload two audio files and view the results. The sample audio files are provided in `qamod/res/audio` directory.

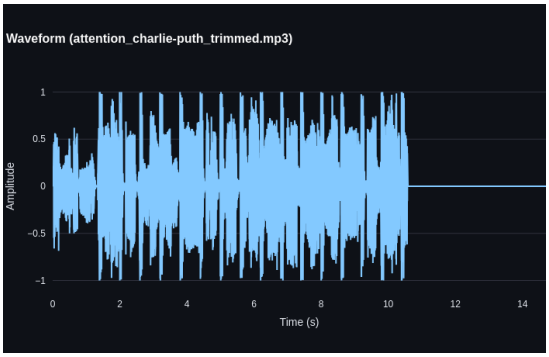
Chapter 4

Results

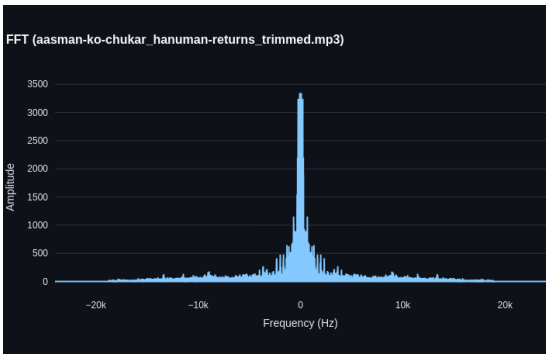
4.1 Input Signals



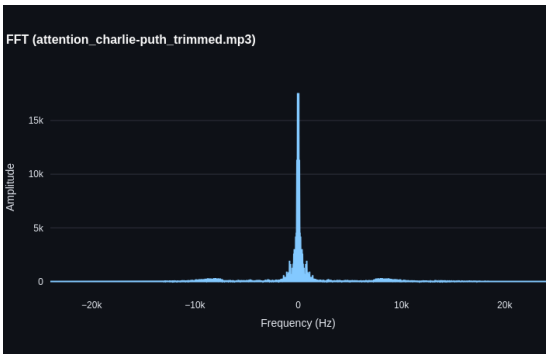
(a) First Waveform



(b) Second Waveform



(c) First Waveform Fourier



(d) Second Waveform Fourier

Figure 4.1: The Input Signals and their Fourier Transforms

4.2 Modulated Signal

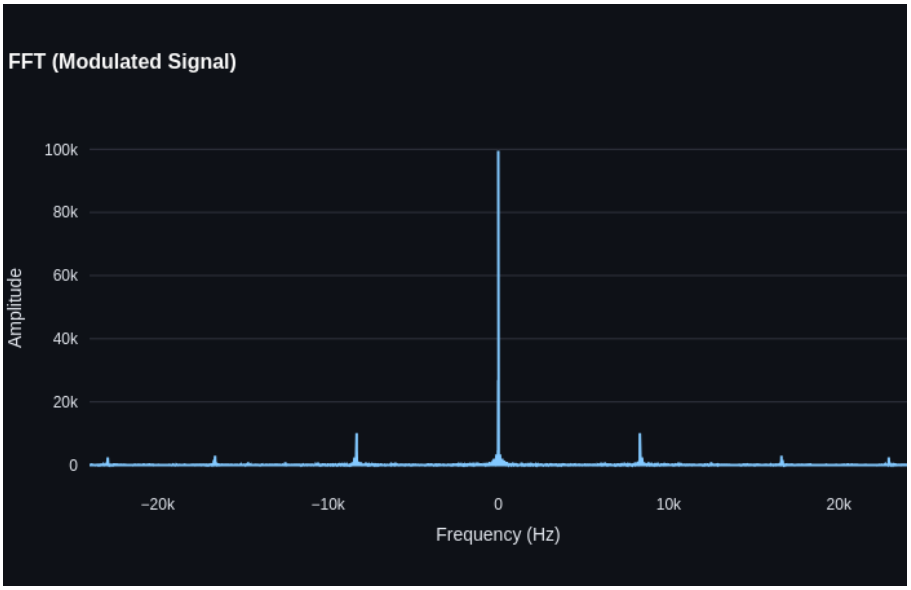
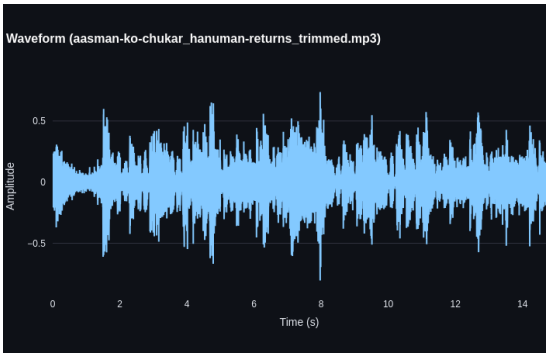
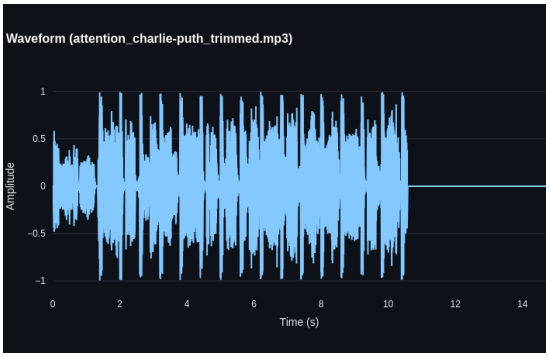


Figure 4.2: Modulated Signal Fourier Trasform

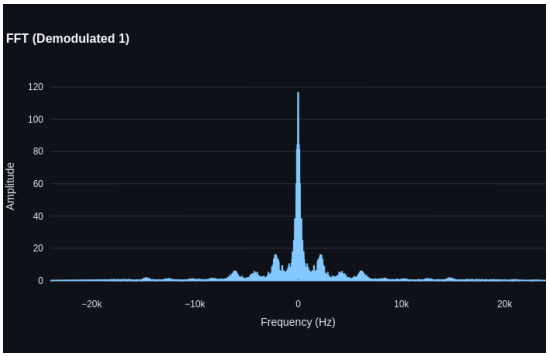
4.3 Ouptut



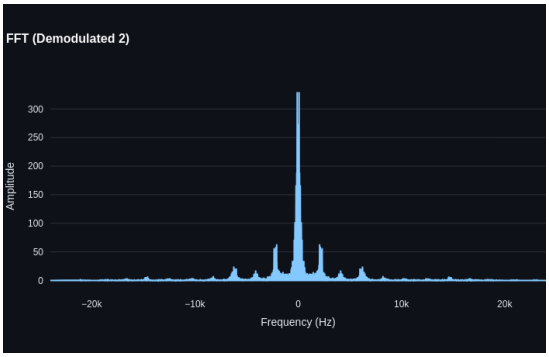
(a) First Waveform



(b) Second Waveform



(c) First Waveform Fourier



(d) Second Waveform Fourier

Figure 4.3: The Output Signals and their Fourier Transforms

Chapter 5

Conclusion

The objectives completed in the project are as follows:

1. The two waveforms and their corresponding Fourier Transforms were displayed
2. The message signals were modulated with Quadrature Amplitude Multiplexing technique
3. The resulting modulated signal's Fourier Transform was also displayed
4. The modulated signal was demodulated and its Fourier Transform and corresponding waveforms obtained, were displayed.
5. The demodulated signals were stored to `.wav` format and made audible

Thus the QAM Modulation technique was successfully studied, understood and implemented in Python.

In conclusion, this project successfully demonstrated the principles of Quadrature Amplitude Modulation (QAM) by implementing both modulation and demodulation techniques. The practical simulations and real-world signal analysis provided deeper insights into QAM's efficiency and robustness in digital communication systems, highlighting its significance in modern telecommunications.

Bibliography

[1] Nikolaos Voudoukis. Performance analysis, characteristics, and simulation of digital qam. *European Journal of Electrical Engineering and Computer Science*, 1, 09 2017.