Text

Description automatically generated with medium confidenceDigital Communication Systems

**Laboratory Report**

Fall 2021

|  |  |
| --- | --- |
| Laboratory Number: | **07** |
| Laboratory Title: | **Quadrature Amplitude Modulation** |
| Full Name: | **Robert Louis Bara** |
| TUID: | **915614617** |

**Description:**

Quadrature Amplitude Modulation is a method of encoding in which the amplitude and phase will carry information regarding the transmitted signal. The symbol set of the QAM encoding can be read as follows, for example a 16QAM splits the bitstream into In-Phase (even indices of the binary bitstream) and Quadrature (odd indices of the bitstream) and encodes the data by calculating the amplitude and Phase angle,

Chart

Description automatically generated with low confidence

Figure . 16QAM symbol set example

The baseband signal, which is the Fourier Transform of the original signal, is then modulated with a carrier signal oscillating at a larger sampling frequency. The received signal is then demodulated by using additive gaussian noise and should retain the transmitted signal’s information. This can be seen by examining the power spectrum density graph’s peaks for each signal.

**Images:**

Chart, scatter chart

Description automatically generated

Figure . Scatter plot

Diagram

Description automatically generated

Figure . Constellation is tilted Scatter Plot, rotated 90 degrees

A picture containing text

Description automatically generated

Figure . Equalizer to transmit TUID

Text

Description automatically generated with medium confidence

Figure . Thresholds to transmit TUID

Transmitting my TUID

Chart, scatter chart

Description automatically generated

Figure . Transmitting TU ID scatter plot

Chart

Description automatically generated with medium confidence

Figure . Constellation after receiving TUID

A picture containing chart

Description automatically generated

Figure . Error when SNR=+inf

The baseband signal retains the fundamental frequency of the signal which is based upon my TU ID(8) +1Hz=1+1=2Hz.

Graphical user interface

Description automatically generated

Figure . Baseband signal

The carrier signal utilizes my TUID (7)+20\*rb, where rb is the fundamental frequency of the signal, therefore we expect a carrier frequency of (6+20)\*2=52Hz.

A screenshot of a computer

Description automatically generated

Figure . Carrier Signal

The modulated signal then samples by multiplying the carrier frequency by 100, and upon demodulation we see approximately the same frequency of about 52Hz.

Graphical user interface

Description automatically generated

Figure . Modulated Signal

With additive noise, the outputted signal retains the information of the original signal.

A screenshot of a computer

Description automatically generated with medium confidence

Figure . Noisy Modulated Signal

**Numerical Tables:**

Examining the signal to noise ratio and error rate within the 16-QAM:

|  |  |
| --- | --- |
| 16-QAM | |
| SNR | Mean Absolute Error |
| 20 | 0 |
| 8 | 0 |
| 0 | 0 |
| -8 | 0.0020 |
| -16 | 0.0490 |
| -20 | 0.1910 |
| -24 | 0.4310 |

**Code:**

### Section 01

The initial parameters are usually defined at the beginning of the program.

clc; clear;

%TU ID: 915614617

A = 1; % Signal amplitude

rb = 2; % Fundamental frequency of signal

Tb = 1 / rb; % Period of signal

fc = (6+20) \* rb;

Tc = 1 / fc;

fs = 100 \* fc; % Sampling frequency

Ts = 1 / fs; % Sampling period

### Section 03

#### Quadrature Amplitude Modulation (QAM)

Suppose there are 16 symbols in the symbol set. Therefore, at least 4 bits are needed to encode the symbols in binary format.

M = 16; % Number of symbols

Nb = ceil(log2(M)); % Number of bits per symbol

symbol\_set = 1:M; % symbols' indices

%symbol\_coords = A \* [1+1j, -1+1j, -1-1j, 1-1j, 3+1j, 3+3j, 1+3j, -1+3j, -3+3j, -3+1j, -3-1j, -3-3j, -1-3j, 1-3j, 3-3j, 3-1j]; % symbols' phase

%figure();

%symbol\_coords = A \* [1+1j, -1+1j, -1-1j, 1-1j, 3+1j, 3+3j, 1+3j, -1+3j, -3+3j, -3+1j, -3-1j, -3-3j, -1-3j, 1-3j, 3-3j, 3-1j]; % symbols' phase

TU5=1/10; %TUID (5)/10

TU6=4/10; %TUID (6)/10

symbol\_coords=A\*[((1+TU5)+(1+TU6)\*1j),((-1-TU5)+(-1+TU6)\*1j),((-1-TU5)+(-1-TU6)\*1j),((1+TU5)+(-1-TU6)\*1j),((3+TU5)+(1+TU6)\*1j),((3+TU5)+(3+TU6)\*1j),((1+TU5)+(3+TU6)\*1j),((-1-TU5)+(3+TU6)\*1j),((-3-TU5)+(3+TU6)\*1j),((-3-TU5)+(1+TU6)\*1j),(-3-TU5)+(-1-TU6\*1j),((-3-TU5)+(-3-TU6)\*1j),((-1-TU5)+(-3-TU6)\*1j),((1+TU5)+(-3-TU6)\*1j),((3+TU5)+(-3-TU6)\*1j),((3+TU5)+(-1-TU6)\*1j)]

figure();

scatter(real(symbol\_coords), imag(symbol\_coords))

set(gca, 'XAxisLocation', 'origin', 'YAxisLocation', 'origin')

xlabel('In-Phase'); ylabel('Quadrature');

constdiag = comm.ConstellationDiagram;

constdiag.Title = "baseband constellation";

% constdiag.SymbolsToDisplay = 10000;

constdiag.ShowLegend = false;

constdiag.ReferenceConstellation = symbol\_coords';

constdiag(symbol\_coords');

constdiag.release();

%msg = 1:M; % message: a set of symbols

msg=[9 1 5 6 1 4 6 1 7]

msg\_coords = symbol\_coords(symbol\_set(msg)); % mepping the symbol to the phase

t = (0:length(msg)-1)\*Tb;

figure()

subplot(3, 1, 1); stem(t, msg); xlabel('time'); ylabel('Symbol Indices'); title('symbols indices');

subplot(3, 1, 2); stem(t, real(msg\_coords)); xlabel('time'); ylabel('In-Phase'); title('symbols in-phase component');

subplot(3, 1, 3); stem(t, imag(msg\_coords)); xlabel('time'); ylabel('Quadrature'); title('symbols quadrature component');

msg\_coords\_fs = rate\_transition(msg\_coords, rb, fs);

t = (0:length(msg\_coords\_fs)-1)\*Ts;

signal = A \* real(msg\_coords\_fs) .\* cos(2 \* pi \* rb \* t) + A \* imag(msg\_coords\_fs) .\* sin(2 \* pi \* rb \* t);

figure()

subplot(3, 1, 1); plot(t, real(msg\_coords\_fs)); xlabel('time'); ylabel('amplitude'); ylim([min(real(msg\_coords\_fs))-0.2, max(real(msg\_coords\_fs))+0.2]); title('Symbols In-Phase')

subplot(3, 1, 2); plot(t, imag(msg\_coords\_fs)); xlabel('time'); ylabel('amplitude'); ylim([min(imag(msg\_coords\_fs))-0.2, max(imag(msg\_coords\_fs))+0.2]); title('Symbols Quadrature')

subplot(3, 1, 3); plot(t, signal); xlabel('time'); ylabel('amplitude'); ylim([min(signal)-0.1, max(signal)+0.1]); title('Baseband Signal');

The baseband signal is modulated with carrier signal and the spectrum will shift in frequency domain from around the origin to the  Hertz.

carrier = 1 \* sin(2 \* pi \* fc \* t + 0);

modulated = signal .\* carrier;

figure()

subplot(2, 1, 1); plot(t, carrier); xlabel('time'); ylabel('amplitude'); ylim([-A-0.1, +A+0.1]); title('carrier');

subplot(2, 1, 2); plot(t, modulated); xlabel('time'); ylabel('amplitude'); ylim([min(signal)-0.1, max(signal)+0.1]); title('Phase Shift Keying (PSK)');

### Section 04

#### Demodulation, Decoding, and Detection

Demodulation is the inverse of modulation, but the operation is the same

% demodulation

rt = modulated;

Ns = round(fs / fc);

demodulated = zeros([1, floor(length(rt) \* fc/fs)]);

dcounter = 1;

for n = Ns:Ns:length(rt)

r\_hat\_t = rt(n-Ns+1:n) .\* sin(2\*pi\*fc\*(0:Ns-1)\*Ts);

demodulated(dcounter) = sum(r\_hat\_t)\*Ts\*2/Tc;

dcounter = dcounter + 1;

end

figure()

plot(demodulated); xlabel('time'); ylabel('amplitude'); title('demodulated signal');

### Section 05

#### PSK Correlator Decoder and Detection

Since sinusoids with different phases have been used for encoding, there are two orthogonal functions and two correlators are needed for detection.

% decoding

rt = demodulated;

Ns = round(fc / rb);

a1 = zeros([1, floor(length(rt) \* rb / fc)]);

a2 = zeros([1, floor(length(rt) \* rb / fc)]);

dcounter = 1;

for n = Ns:Ns:length(rt)

% computing a1 (quadrature)

r\_hat\_t = rt(n-Ns+1:n) .\* sin(2\*pi\* (fc/rb) \* (0:Ns-1)/(Ns-1) \* Tc/Tb);

a1(dcounter) = sum(r\_hat\_t)\*(Tc/Tb)\*sqrt(2)\*2;

% computing a2 (in-phase)

% r\_hat\_t = rt(n-Ns+1:n) .\* sin(2\*pi\* (fc/rb) \* (0:Ns-1)/(Ns-1) \* Tc/Tb + pi/2);

r\_hat\_t = rt(n-Ns+1:n) .\* cos(2\*pi\* (fc/rb) \* (0:Ns-1)/(Ns-1) \* Tc/Tb);

a2(dcounter) = sum(r\_hat\_t)\*(Tc/Tb)\*sqrt(2)\*2;

dcounter = dcounter + 1;

end

figure()

subplot(2, 1, 1); stem(a1); xlabel('symbols index'); ylabel('amplitude'); title('detected symbols phase (a1)');

subplot(2, 1, 2); stem(a2); xlabel('symbols index'); ylabel('amplitude'); title('detected symbols phase (a2)');

figure(); scatter(a1, a2);

set(gca, 'XAxisLocation', 'origin', 'YAxisLocation', 'origin');

xlabel('quadrature'); ylabel('in-phase');

% detection

%equalizer = [a1(1), a2(1)];

%centers\_complex = a1 / equalizer(1) + 1j \* a2 / equalizer(2)

equalizer = [1.9316,1.5790];

centers\_complex =[1.00000000000000 + 1.00000000000000i, -0.428571428571429 - 1.00000000000000i, -1.00000000000000 - 0.999999999999999i, -0.999999999999999 + 1.00000000000000i, 1.00000000000000 + 2.81818181818182i, 2.42857142857143 + 2.81818181818181i, 2.42857142857143 + 0.999999999999992i, 2.42857142857142 - 1.00000000000001i, 2.42857142857142 - 2.81818181818183i, 0.999999999999992 - 2.81818181818183i, -0.285714285714301 - 3.72727272727273i, -2.42857142857144 - 2.81818181818180i, -2.42857142857143 - 0.999999999999984i, -2.42857142857142 + 1.00000000000002i, -2.42857142857141 + 2.81818181818184i, -0.999999999999984 + 2.81818181818183i]

decoding\_table = 1:M;

detected\_indices = zeros([1, length(a1)]);

for n = 1:length(a1)

distances = abs(centers\_complex - (a1(n) / equalizer(1) + 1j \* a2(n) / equalizer(2)));

detected\_indices(n) = decoding\_table(distances == min(min(distances)));

end

figure()

stem(detected\_indices); xlabel('symbols index in the message stream'); ylabel('symbols index'); title('detected symbols indices');

% computing error rates

err = mean(msg ~= detected\_indices)

### Section 06

#### Noise

Noise is the impact of channel that signal transfer through.

The most common way to find the noise impact is Additive White Gaussian Noise (AWGN) model. In this model, the noise will be generted by a Gaussian distribution and added to the transmitted signal.

% generation of a random message with fixed length

rng(0);

Nm = 1000;

msg = randi(M, [1, Nm]); % message: a set of symbols

figure(); stem(msg(1:20));

msg\_coords = symbol\_coords(symbol\_set(msg)); % mepping the symbol to the amplitude

msg\_coords\_fs = rate\_transition(msg\_coords, rb, fs); % translate rate 1 to rate rb

t = (0:length(msg\_coords\_fs)-1)\*Ts; % make the parameter time

signal = A \* real(msg\_coords\_fs) .\* cos(2 \* pi \* rb \* t) + A \* imag(msg\_coords\_fs) .\* sin(2 \* pi \* rb \* t); % baseband signal

figure(); plot(signal(1:10000));

carrier = 1 \* sin(2 \* pi \* fc \* t + 0);

modulated = signal .\* carrier;

figure()

subplot(2, 1, 1); plot(t(1:5000), carrier(1:5000)); xlabel('time'); ylabel('amplitude'); ylim([-A-0.1, +A+0.1]); title('carrier');

subplot(2, 1, 2); plot(t(1:5000), modulated(1:5000)); xlabel('time'); ylabel('amplitude'); ylim([-A-0.1, +A+0.1]); title('Quadrature Amplitude Modulation (QAM)');

snr = 0;

noisy\_signal = awgn(modulated, snr, 'measured');

figure(); plot(noisy\_signal(1:5000));

% demodulation

rt = noisy\_signal;

Ns = round(fs / fc);

demodulated = zeros([1, floor(length(rt) \* fc/fs)]);

dcounter = 1;

for n = Ns:Ns:length(rt)

r\_hat\_t = rt(n-Ns+1:n) .\* sin(2\*pi\*fc\*(0:Ns-1)\*Ts);

demodulated(dcounter) = sum(r\_hat\_t)\*Ts\*2/Tc;

dcounter = dcounter + 1;

end

figure()

plot(demodulated(1:200)); xlabel('time'); ylabel('amplitude'); title('demodulated signal');

% decoding

rt = demodulated;

Ns = round(fc / rb);

a1 = zeros([1, floor(length(rt) \* rb / fc)]);

a2 = zeros([1, floor(length(rt) \* rb / fc)]);

dcounter = 1;

for n = Ns:Ns:length(rt)

% computing a1 (quadrature)

r\_hat\_t = rt(n-Ns+1:n) .\* sin(2\*pi\* (fc/rb) \* (0:Ns-1)/(Ns-1) \* Tc/Tb);

a1(dcounter) = sum(r\_hat\_t)\*(Tc/Tb)\*sqrt(2)\*2;

% computing a2 (in-phase)

% r\_hat\_t = rt(n-Ns+1:n) .\* sin(2\*pi\* (fc/rb) \* (0:Ns-1)/(Ns-1) \* Tc/Tb + pi/2);

r\_hat\_t = rt(n-Ns+1:n) .\* cos(2\*pi\* (fc/rb) \* (0:Ns-1)/(Ns-1) \* Tc/Tb);

a2(dcounter) = sum(r\_hat\_t)\*(Tc/Tb)\*sqrt(2)\*2;

dcounter = dcounter + 1;

end

figure()

subplot(2, 1, 1); stem(a1(1:20)); xlabel('symbols index'); ylabel('amplitude'); title('detected symbols in-phase (a1)');

subplot(2, 1, 2); stem(a2(1:20)); xlabel('symbols index'); ylabel('amplitude'); title('detected symbols quadrature (a2)');

detected\_indices = zeros([1, length(a1)]);

for n = 1:length(a1)

distances = abs(centers\_complex - (a1(n) / equalizer(1) + 1j \* a2(n) / equalizer(2)));

detected\_indices(n) = decoding\_table(distances == min(min(distances)));

end

figure()

stem(detected\_indices(1:20)); xlabel('symbols index in the message stream'); ylabel('symbols index'); title('detected symbols indices');

decoded\_complex = a1 / equalizer(1) + 1j \* a2 / equalizer(2);

constdiag = comm.ConstellationDiagram;

constdiag.Title = "received baseband constellation";

% constdiag.SymbolsToDisplay = 10000;

constdiag.ShowLegend = false;

constdiag.ReferenceConstellation = msg\_coords';

constdiag(decoded\_complex');

constdiag.release();

% computing error rates

err = mean(msg ~= detected\_indices)

### Section 07

Power spectrum of the baseband signal:

scope1 = dsp.SpectrumAnalyzer();

scope1.SampleRate = fs;

scope1.PlotAsTwoSidedSpectrum = false;

scope1.SpectrumUnits = "dBW";

scope1(signal');

scope1.Name = 'Baseband Signal';

release(scope1);

Power spectrum of the carrier signal:

scope2 = dsp.SpectrumAnalyzer();

scope2.SampleRate = fs;

scope2.PlotAsTwoSidedSpectrum = false;

scope2.SpectrumUnits = "dBW";

scope2(carrier');

scope2.Name = 'Carrier';

release(scope2);

Power spectrum of the modulated (passband) signal:

scope3 = dsp.SpectrumAnalyzer();

scope3.SampleRate = fs;

scope3.PlotAsTwoSidedSpectrum = false;

scope3.SpectrumUnits = "dBW";

scope3(modulated');

scope3.Name = 'Modulated Signal';

release(scope3);

Power spectrum of the recieved signal:

scope4 = dsp.SpectrumAnalyzer();

scope4.SampleRate = fs;

scope4.PlotAsTwoSidedSpectrum = false;

scope4.SpectrumUnits = "dBW";

scope4(noisy\_signal');

scope4.Name = 'Noisy Modulated Signal';

release(scope4);

### Functions Definition

Rate transition:

function rt = rate\_transition(s, f1, f2)

if (f1 < f2)

rt = repelem(s, floor(f2/f1));

% x = 0:1/f1:((length(s)-1)\*1/f1);

% xq = 0:1/f2:((length(s)-1)\*1/f1);

% rt = interp1(x, s, xq, 'linear');

else

step = floor(f1/f2);

rt = s(1:step:end);

end

end