*THE REAL SCIENCE

Lab 4. Modulation. Data transfer. Error control.

Department of Computer Science

Student: Patrushev Boris 19213

Advisor: Ruslan V. Akhpashev

Date: 26/05/2021

Variant: 9. QAM16

Implement modulation function (QAM16) based on formulas in 3GPP TS38.211 specification (clause 5). Correspondingly encoder and decoder.

In [1]:

```
import numpy as np


# encode by the formula from the specification
def encode(b0, b1, b2, b3):
    return \
        np.complex128(
            np.complex128(
                (1 - 2 * b0) * (1 + 2 * b2)
                + (1 - 2 * b1) * (1 + 2 * b3)
                * np.sqrt(np.complex128(-1))
            ) / 4
        )


# decode to bits
def decode(num: np.complex_):
    b0 = (num.real < 0) * 1
    b1 = (num.imag < 0) * 1
    b2 = (abs(num.real) > 0.5) * 1
    b3 = (abs(num.imag) > 0.5) * 1
```

```
    return b0, b1, b2, b3


# split a byte in half and encode it
def split_encode_byte(byte):
    return (encode((byte & 0x80) >> 7,
                   (byte & 0x40) >> 6,
                   (byte & 0x20) >> 5,
                   (byte & 0x10) >> 4
                   ),
            encode((byte & 0x08) >> 3,
                   (byte & 0x04) >> 2,
                   (byte & 0x02) >> 1,
                   (byte & 0x01))
            )


# glue two half-bytes to a byte
def glue_to_byte(hb1, hb2):
    return hb1[0] * 128 \
        + hb1[1] * 64 \
        + hb1[2] * 32 \
        + hb1[3] * 16 \
        + hb2[0] * 8 \
        + hb2[1] * 4 \
        + hb2[2] * 2 \
        + hb2[3]
```

The encode function converts 4 data bits to a complex number, decode does exactly the opposite, and the split_encode_byte and glue_to_byte functions respectively split the data byte and encode it and glue the two halves of the byte into one.

Next, we implement the server The server takes the file, encodes it, and sends it to each client.

Develop a client-server application.(https://github.com/fzybot/simpleClientServer)

In [ ]:

```
import pickle
import socket
import time
import qam16


# gets data and encodes it
def get_data_for_sending() -> [complex]:
```

```python
    # read raw data from file
    with open("in.data", "rb") as file:
        file_contents = file.read(-1)
    ret = []
    for byte in file_contents:
        # split the byte in half and encode them to complex values
        hb1, hb2 = qam16.split_encode_byte(byte)

        # pickle the complex numbers and append to return value
        ret.append(hb1)
        ret.append(hb2)

    print("data ready to be sent!")
    return ret


print("preparing data...")
data = get_data_for_sending()

# pickle data
bytes_to_send = b''.join(list(map(lambda a: pickle.dumps(a), data)))
bytes_to_send += b''


SERVER_ADDRESS = ('localhost', 8686)

# open socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
while True:
    try:
        server_socket.bind(SERVER_ADDRESS)
    except OSError:
        time.sleep(5)
        continue
    break
server_socket.listen(10)
print('server is running, press ctrl+c to stop')

# for each connection, send the data
while True:
    connection, address = server_socket.accept()
    print("new connection from {address}".format(address=address))

    connection.send(bytes_to_send)
```

```
    print("data sent!")
    connection.close()
    print("connection closed!")
```

```
preparing data...
data ready to be sent!
server is running, press ctrl+c to stop
```

Next, let's make a client:

In [ ]:

```
import pickle
import sys
from typing import IO
import qam16
import io
import socket

# we'll need it for BER calculation
with open("in.data", "rb") as ogfile:
    ogdata = ogfile.read(-1)
data_len = len(ogdata)

# a lookup table for count of bits in a byte
# needed for BER calculation
bit_lookup_table = [
    sum(
        [
         1 if x & (1 << i)
         else 0
         for i in range(8)
         ]
    )
    for x in range(256)
]


# unpickle multiple objects and generate them one by one
def load(file: IO[bytes]) -> [complex]:
    while True:
        try:
            yield pickle.load(file)
        except EOFError:
            break
```

```python
MAX_CONNECTIONS = 20
address_to_server = ('localhost', 8686)

clients = [socket.socket(socket.AF_INET, socket.SOCK_STREAM) for i in range(MAX_CONNECTIONS)]

SNR_list = [-6 + i for i in range(MAX_CONNECTIONS)]

for n in range(len(clients)):
    clients[n].connect(address_to_server)
    print("connected!")

    # receiving the pickled complex numbers
    data = []
    while True:
        pack = clients[n].recv(1024)
        if not pack:
            break
        data.append(pack)
    print("data received!")
    data = b''.join(data)
    # now we unpickle and decode all received data
    halfbyte_list = []
    with io.BytesIO(data) as f:
        for i in load(f):
            halfbyte_list.append(qam16.decode(qam16.apply_noise(i, SNR_list[n])))

    print("decoded data!")

    # glue the halfbytes and put them in a list
    byte_list = []
    for i in range(len(halfbyte_list) // 2):
        byte_list.append(
            qam16.glue_to_byte(
                halfbyte_list[2 * i],
                halfbyte_list[2 * i + 1]
            )
        )

    # make them singleton `bytes` objects just in case
    byte_list = [int(i & 0xFF).to_bytes(1, sys.byteorder, signed=False)
                 for i in byte_list]

    print(f"writing to floppas/out{n + 1}.data ...")
```

```
    # write into a file
    with open(f"floppas/out{n + 1}.data", "bw") as file:
        for i in byte_list:
            file.write(i)

    newdata = b''.join(byte_list)
    bit_error_count = 0
    for i in range(data_len):
        # XOR shows the errors bit by bit
        # lookup gets amount of bit errors
        # so for each iteration we add bit errors per byte
        bit_error_count += bit_lookup_table[newdata[i] ^ ogdata[i]]
    with open("BER.txt", 'ta') as file:
        file.write(f"SNR #{n+1}: {bit_error_count / (data_len * 8)}\n")

    print("done!")
```

The client receives the data, adds noise, decodes the noisy data, writes it to a file, then calculates the BER and writes it to the BER.txt file

I also wrote a small program that converts the recorded files back to pictures:

In [2]:

```
from PIL import Image

for i in range(20):
    with open(f"floppas/out{i+1}.data", "rb") as f:
        data = f.read(-1)
    im = Image.frombytes(data=data, size=(320, 276), mode="RGB")
    im.save(format="JPEG", fp=f"{i+1}.jpg")
```

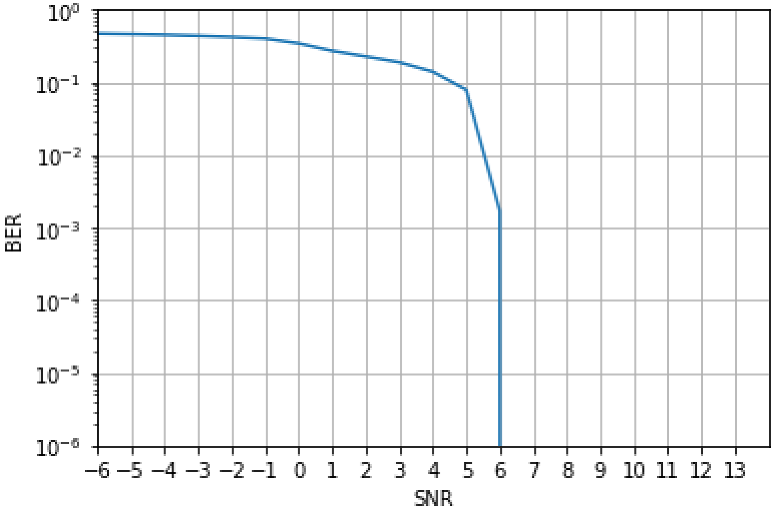
Let's see the BER / SNR graph:

In [3]:

```
import matplotlib.pyplot as plt

arr = [float(i[9:]) for i in open("BER.txt").readlines()]
print(arr)
fig = plt.figure()
plt.plot([i - 6 for i in range(20)], arr)
plt.yscale("log")
plt.axis(ymin=0.1**6, ymax=1, xmin=-6, xmax=14)
plt.xticks([i - 6 for i in range(20)])
plt.ylabel("BER")
```

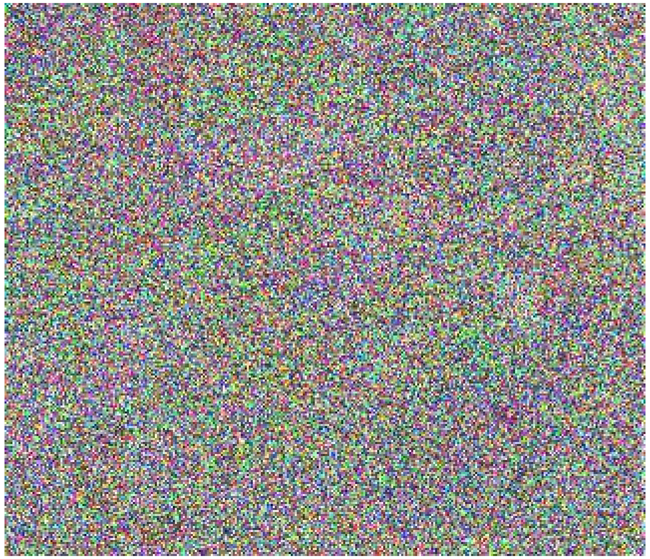
```python
plt.xlabel("SNR")
plt.grid(True)
plt.show()
```

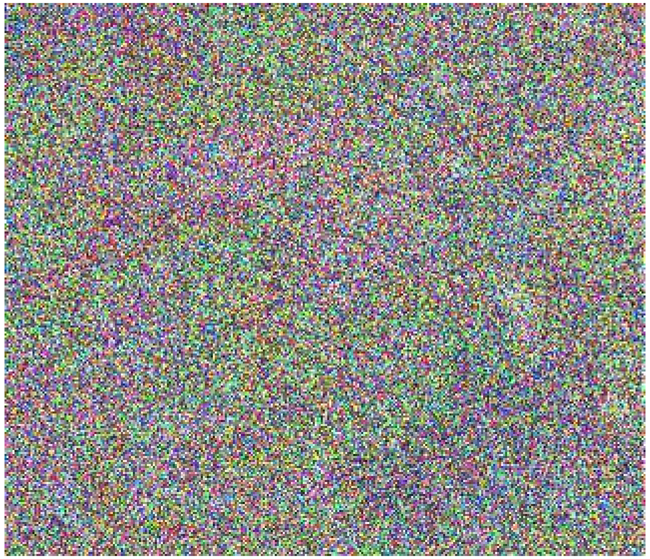
```
[0.469142040307971, 0.4615446671195652, 0.4510968636775362, 0.43773541289251205, 0.4212324501811594, 0.40121763662439613, 0.3438655834842995, 0.27110743131038645, 0.22675686896135266, 0.18862705691425122, 0.13989045516304346, 0.07879727128623189, 0.0017172403381642512, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

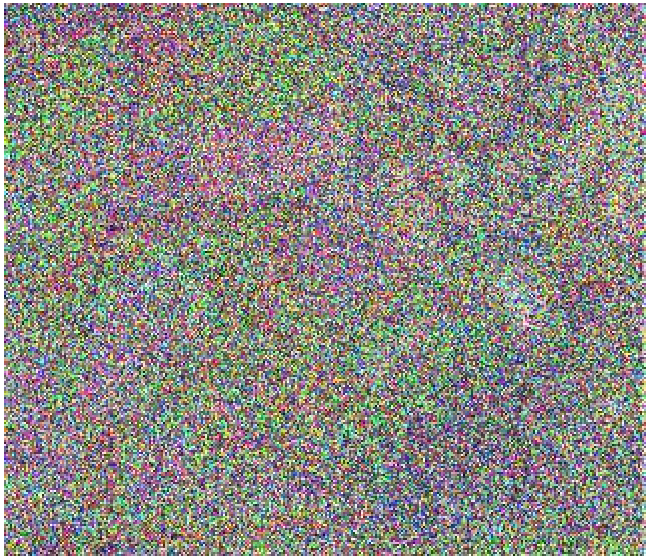


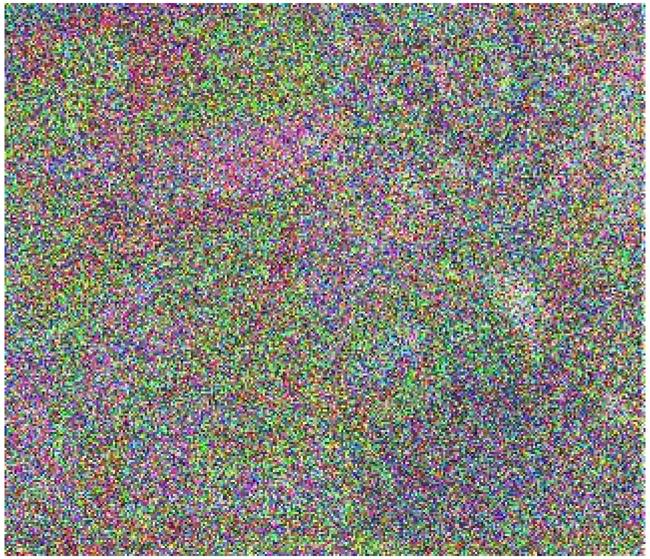
Let's look at the pictures received by clients:

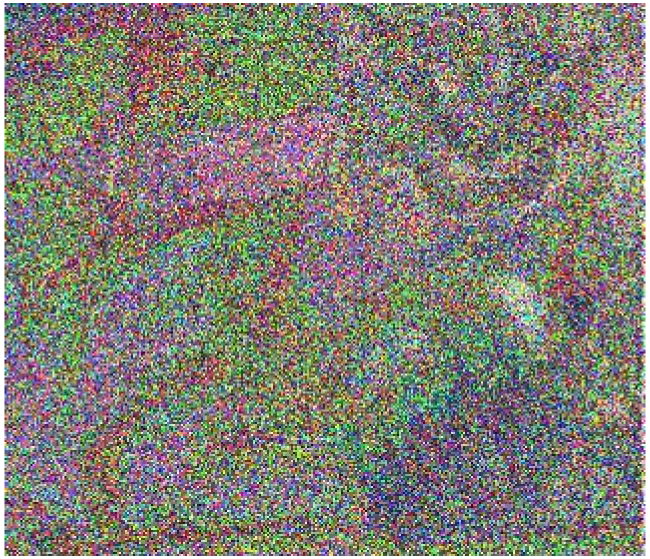
In [4]:
```python
import IPython.display as ipy

img_name_list = [f"jpg_floppas/floppa_{i + 1}.jpg" for i in range(20)]
for i in img_name_list:
    ipy.display(ipy.Image(filename=i))
```

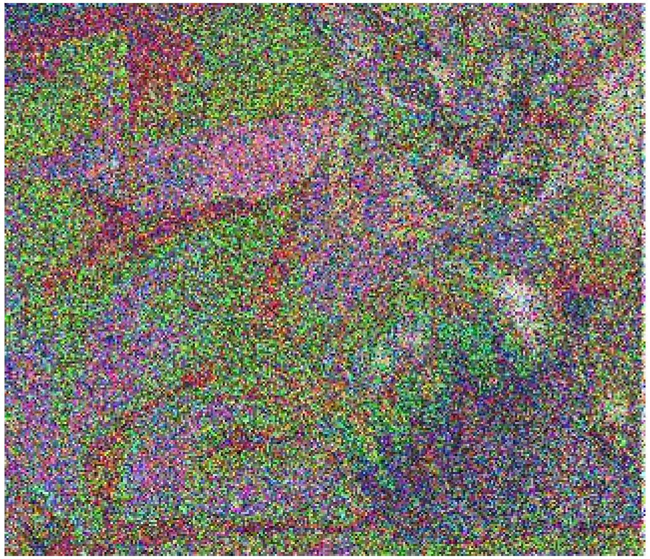


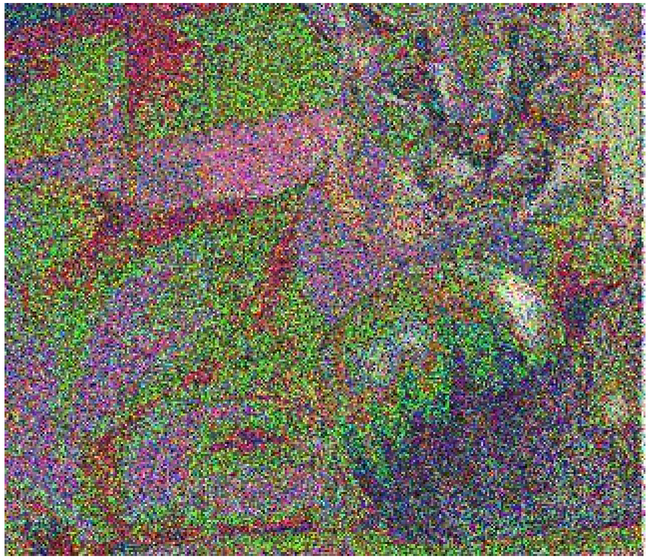


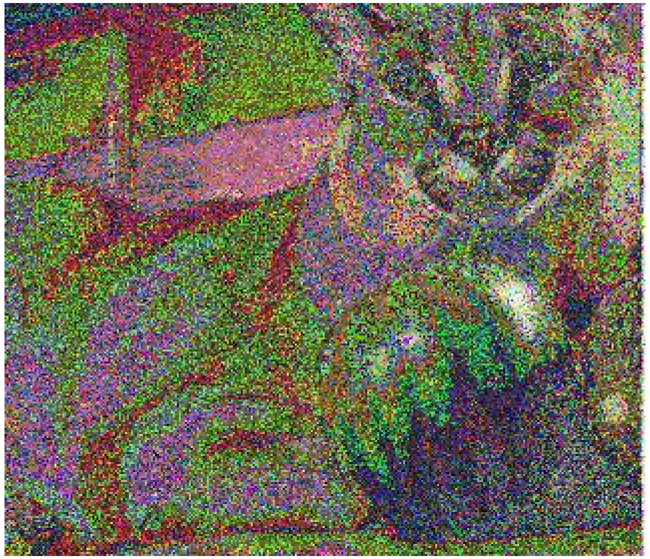


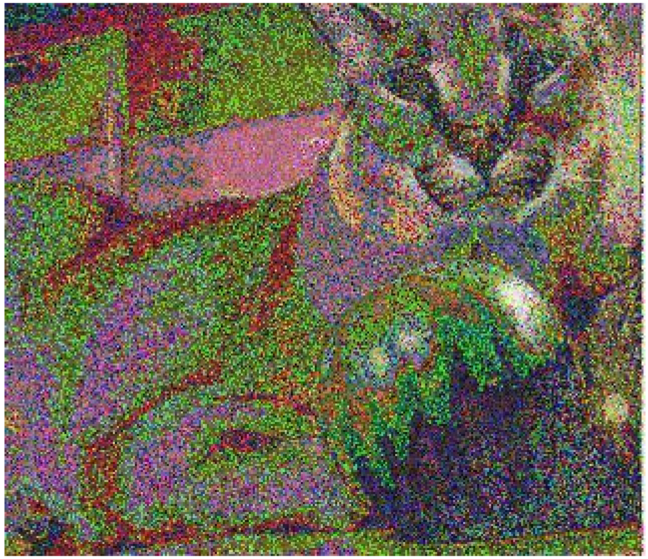
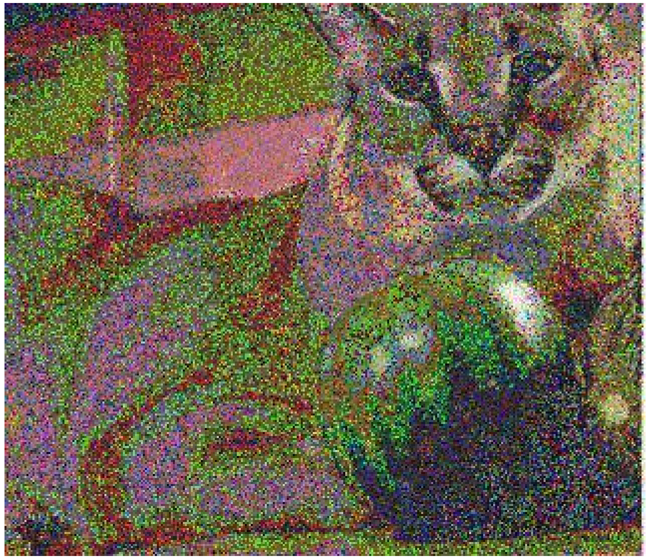


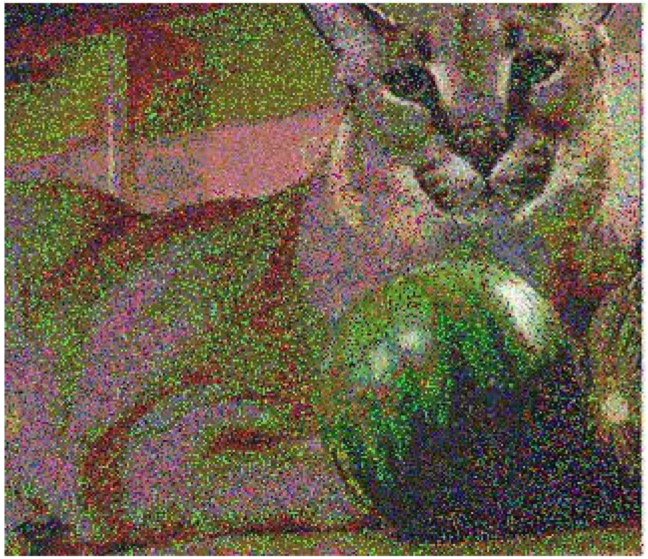


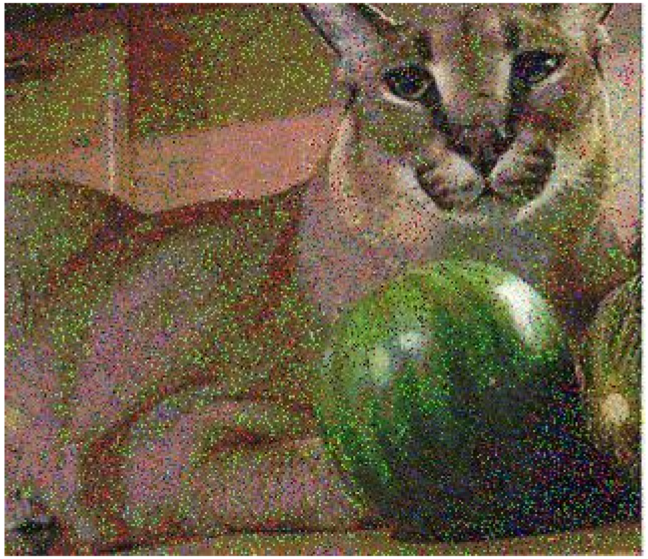













From about the 13th picture, the noise disappears completely, which completely coincides with the theory: SNR = <7dB, which means the values differ by less than 1/4, which means that decoding is successful.

In [ ]: