****

ENG4052: Digital Communication 4 (2022-23)

Lab5: OFDM Communications Link

**Ran Shuai (2633609R)**

*Table Contents*

[*OFDM Transmitter 3*](#_Toc130910199)

[*What I have done 3*](#_Toc130910200)

[*The results 4*](#_Toc130910201)

[*Conclusion 5*](#_Toc130910202)

[*The length and features of complex\_signal 5*](#_Toc130910203)

[*Pilot carriers are uniformly distributed 5*](#_Toc130910204)

[*Power distribution 5*](#_Toc130910205)

[*OFDM Receiver 6*](#_Toc130910206)

[*What I have done 6*](#_Toc130910207)

[*The results 6*](#_Toc130910208)

[*Conclusion 8*](#_Toc130910209)

[*The offset 8*](#_Toc130910210)

[*Bit Error Ratio 8*](#_Toc130910211)

[*Distortion and Noise in the Communications Channel 9*](#_Toc130910212)

[*What I have done 9*](#_Toc130910213)

[*The results 9*](#_Toc130910214)

[*Conclusion 13*](#_Toc130910215)

[*Effect of reverberance and damping 13*](#_Toc130910216)

[*Effect of Additive white noise 13*](#_Toc130910217)

[*Reed-Solomon Channel Coding 14*](#_Toc130910218)

[*What I have done 14*](#_Toc130910219)

[*The results 14*](#_Toc130910220)

[*Conclusion 15*](#_Toc130910221)

[*References 16*](#_Toc130910222)

[*1.2 16*](#_Toc130910223)

[*1.3 18*](#_Toc130910224)

[*1.4 19*](#_Toc130910225)

[*1.5 21*](#_Toc130910226)

# OFDM Transmitter

Orthogonal frequency division multiplexing belongs to multi carrier transmission technology. The so-called multi carrier transmission technology refers to dividing the available spectrum into multiple subcarriers, each of which can carry a low speed data flow.

My understanding is: Unlike the transmission process we learned before, in OFDM we do not use a single frequency to transmit the signal, but use multiple orthogonal frequencies to transmit the signal, with each frequency transmitting a small portion of the signal.

## What I have done

First, I modulated some random complex signals using OFDM, and then plotted the real and imaginary parts of the modulated signal he same graph.

Then I plotted the absolute value of its discrete Fourier transforms without the prefix to observe the energy distribution of OFDM carriers.

Finally, I encoded an 8-bit depth grayscale image

## The results

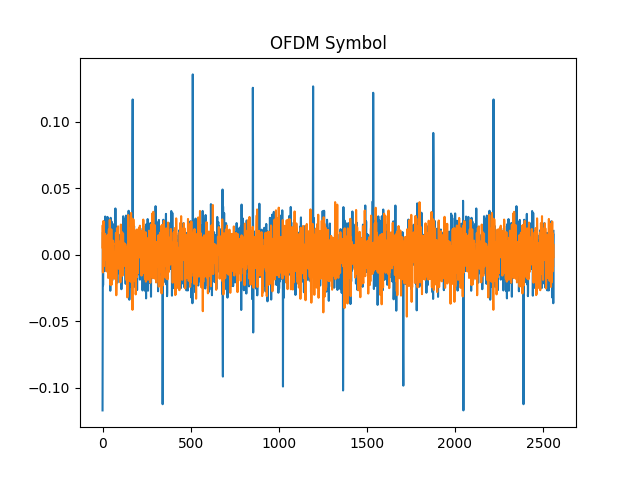


Figure 1.1 : The real and imaginary components of encoded signal

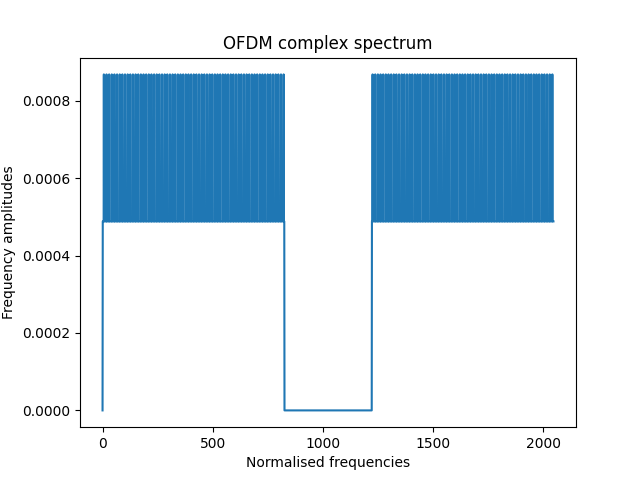


Figure 1.2 : OFDM Spectrum

## Conclusion

### The length and features of complex\_signal

The length of the encoded signal is 2560.

2048 is the number of subcarriers.

It means the length of the cyclic prefix.

### Pilot carriers are uniformly distributed

From Figure 1, we can draw the conclusion below :

The pilot carrier is uniformly distributed in OFDM coding, so on the real part of the OFDM coded signal, we can observe positive and negative peaks at equal intervals, which are generated by the pilot carrier.

### Power distribution

From figure 2, we can see that the lines are relatively flat at the occupied bandwidth. This also indicates that the pilot carrier is uniformly distributed in OFDM coding.

# OFDM Receiver

## What I have done

First, I identified the start of the OFDM symbol and plotted the cross-correlation between the received signal and the OFDM symbol. I also plotted the image of the sum of the magnitudes of the Fourier transform values of the received OFDM symbol. Then, I plotted the decoded image after OFDM demodulation.

## The results

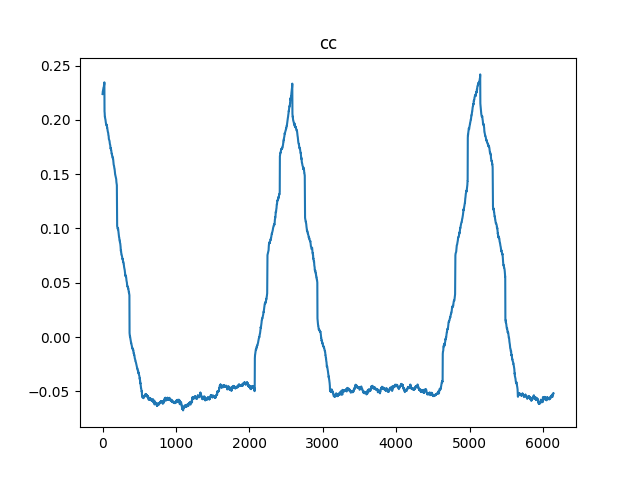


Figure 2.1: Cross correlation image

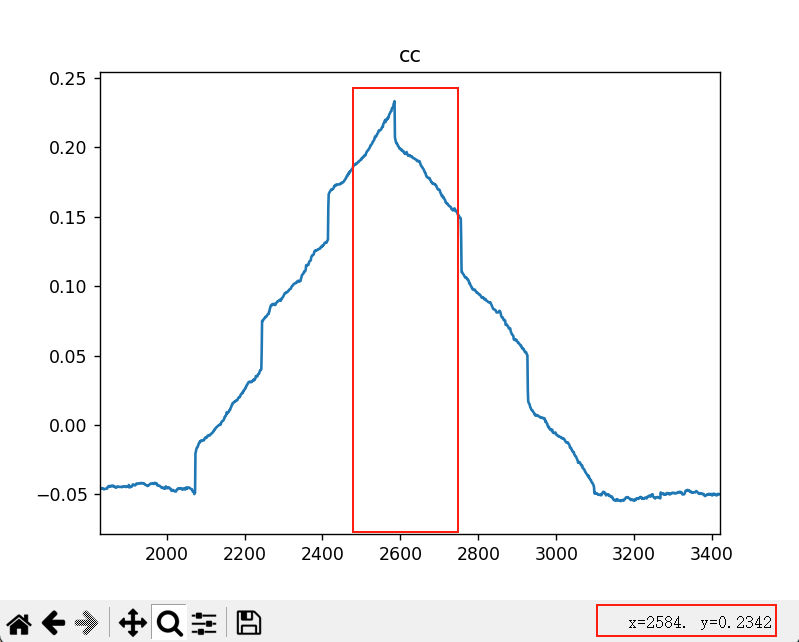


Figure 2.2: Cross correlation image which is zoomed in

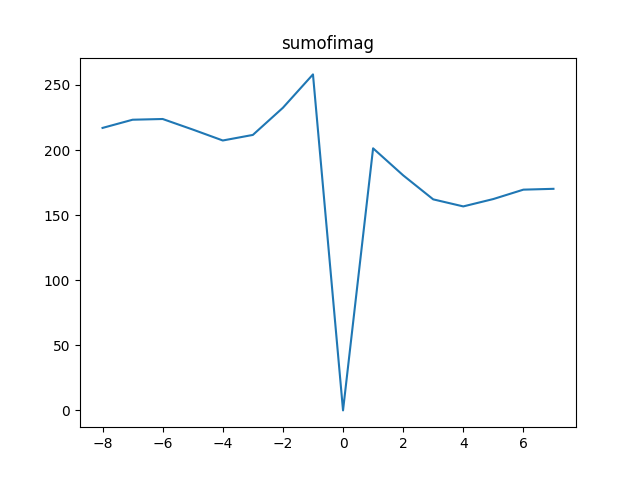


Figure 2.3: Sum of amplitudes

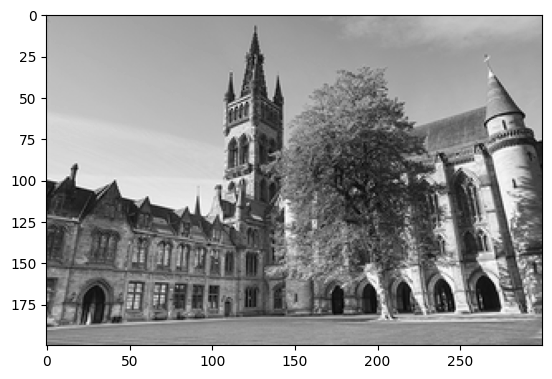


Figure 2.4: The demodulated picture

## Conclusion

### The offset

The value of offset is 2585. However, the length of an OFDM symbol is 2560

So the true value of offset is

And the length of the dummy data I inserted is 50 which is twice 25.

From figure 2.2, we can also see that the offset is 2585 (2560+25). This also implies that 25 is the offset.

### Bit Error Ratio

The bit error ratio is zero. Because we do not add any noise during the transmit signal.

# Distortion and Noise in the Communications Channel

## What I have done

1. I added a reverb effect to the original audio: 50% reverberance and 50% damping. The resulting image is Figure 3.1 and the bit error ratio is 0.878%.
2. Adjust the reverberance to 20% and damping to 50%. The resulting image is Figure 3.2 and the bit error ratio is 0.291%.
3. Adjust the reverberance to 80% and damping to 50%. The resulting image is Figure 3.3 and the bit error ratio is 2.045%.
4. Adjust the reverberance to 50% and damping to 20%. The resulting image is Figure 3.4 and the bit error ratio is 1.303%.
5. Adjust the reverberance to 50% and damping to 80%. The resulting image is Figure 3.5 and the bit error ratio is 0.625%.
6. Additive white noise: Add Noise of 0.05 amplitude so SNR equals to -5.555dB
7. Additive white noise: Add Noise of 0.015 amplitude so SNR equals to 4.9127dB

## The results

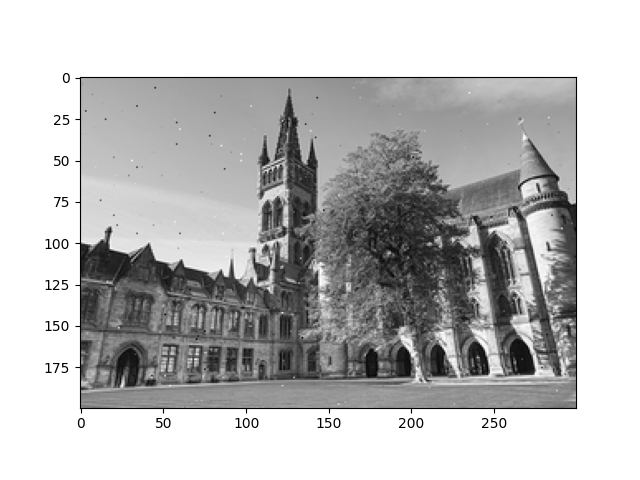


Figure 3.1: Default Setting BER = 0.008783333333333334

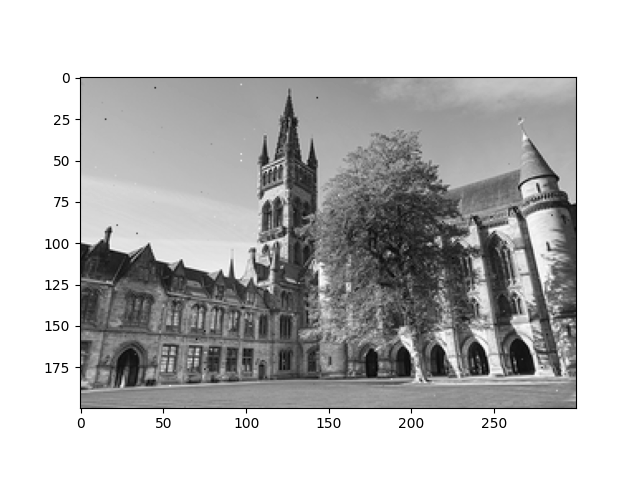


Figure 3.2: Reverberance = 20% BER = 0.002916666666666667

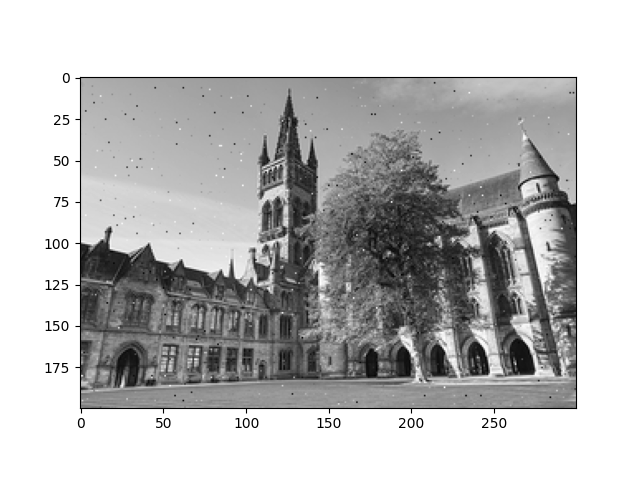


Figure 3.3: Reverberance = 80% BER = 0.02045

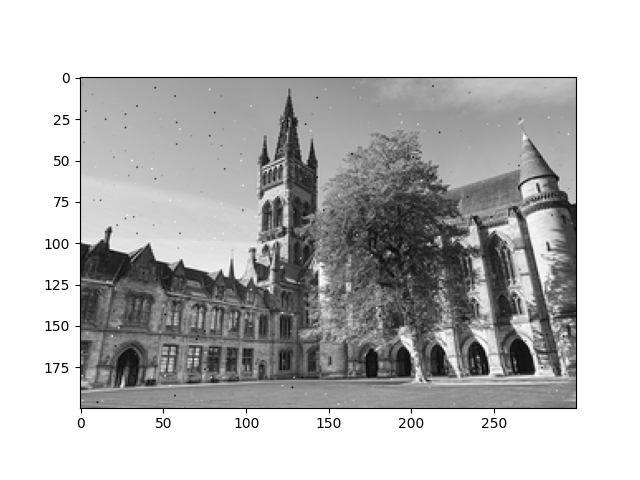


Figure 3.4: Damping = 20% BER = 0.013033333333333333

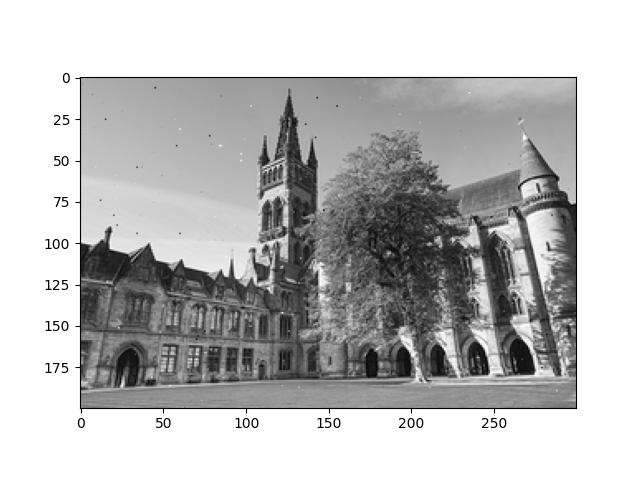


Figure 3.5: Damping= 80% BER = 0.02045

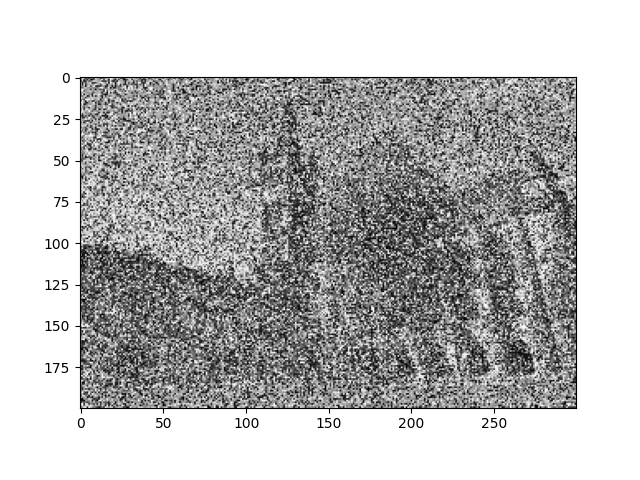


Figure 3.6: SNR = -5.5dB BER = 0.9379666666666666

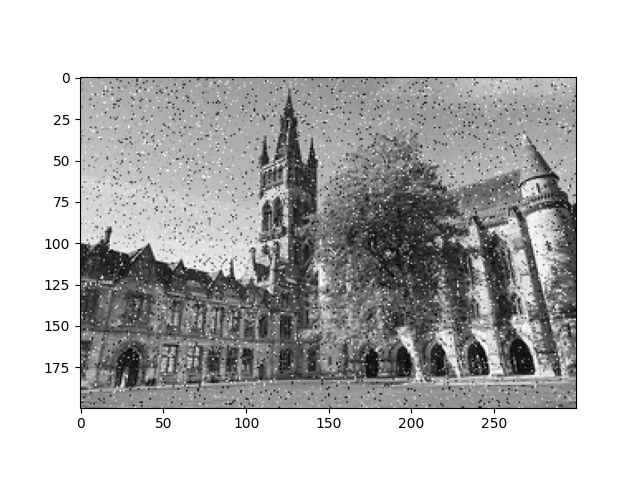


Figure 3.7: SNR = 4.9127dB BER = 0.2511

## Conclusion

### Effect of reverberance and damping

Signal distortion and temporal dispersion will result from increased attenuation of the signal at various frequencies due to increased reverberance. As a result, there is greater signal interference, which raises the bit error rate. The damping parameter regulates how much the system's signal oscillates; the lesser the damping value, the higher the oscillation, which raises the chance of signal interference and, therefore, the bit error rate.

### Effect of Additive white noise

This is because under low signal-to-noise ratios, the quality of the signal is affected by noise, and the receiver may misinterpret the noise as a data signal, or misinterpret the noise in the data signal as other data signals, which leads to an increase in the bit error rate.

# Reed-Solomon Channel Coding

Reed Solomon channel coding is a widely used error correction coding technique

## What I have done

1. Use RSC to encode and decode the image without additive white noise.
2. Compared the BER of figure 3.7 with the BER vs SNR of RSC Picture in lecture

## The results

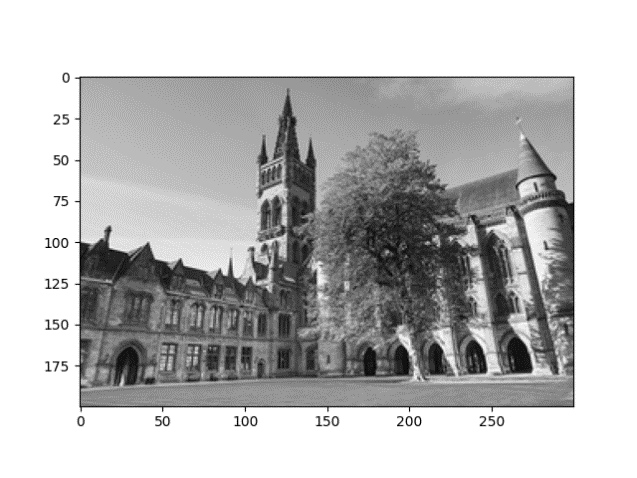


Figure 4.1: Without Additive white noise BER = 0

## Conclusion

In case 1.4, when SNR equals to 4.9127dB, BER equals to 0.2511.

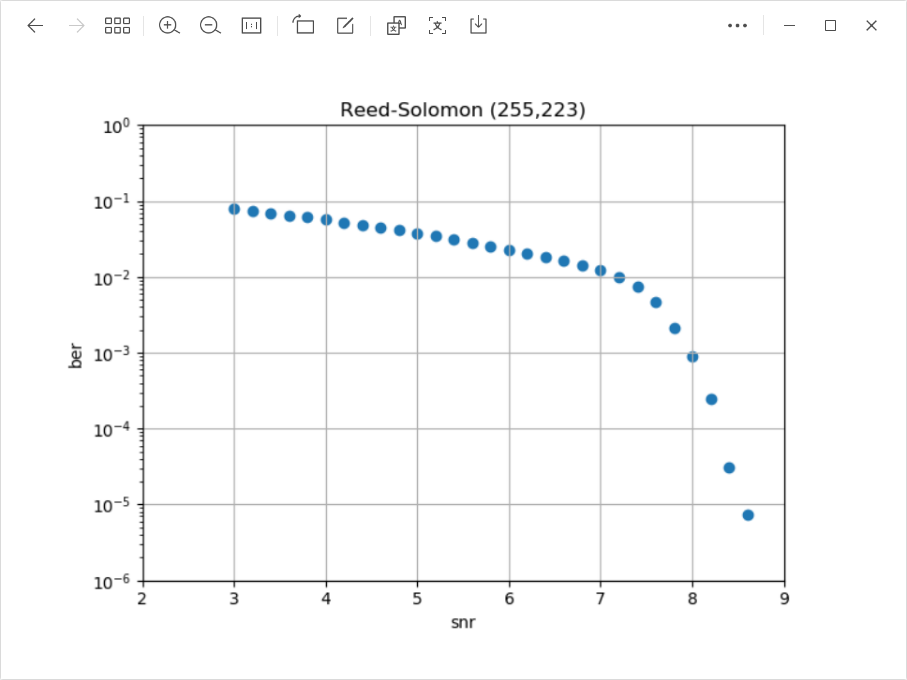


Figure 4.2: BER vs SNR of Reed-Solomon Codes

In Figure 4.2, when the SNR is around 5dB, the BER using RSC encoding should be around 0.05. However, in Figure 3.7, we did not use RSC encoding to modulate and demodulate the image, and the error rate increased by about 5 times to 0.25. This indicates that RSC code can really lower the BER.

# References

## 1.2

from PIL import Image

import numpy as np

import scipy.io.wavfile as wav

import pyofdm.codec

import pyofdm.nyquistmodem

import matplotlib.pyplot as plt

*# Number of total frequency samples*

totalFreqSamples = 2048

*# Number of useful data carriers / frequency samples*

sym\_slots = 1512

*# QAM Order*

QAMorder = 2

*# Total number of bytes per OFDM symbol*

nbytes = sym\_slots \* QAMorder // 8

*# Distance of the evenly spaced pilots*

distanceOfPilots = 12

pilotlist = pyofdm.codec.setpilotindex(nbytes, QAMorder, distanceOfPilots)

ofdm = pyofdm.codec.OFDM(pilotAmplitude=16/9,

                         nData=nbytes,

                         pilotIndices=pilotlist,

                         mQAM=QAMorder,

                         nFreqSamples=totalFreqSamples)

row = np.random.randint(256, size=nbytes, dtype="uint8")

complex\_signal = ofdm.encode(row)

print(complex\_signal)

print("---------------------------------------------")

print(np.real(complex\_signal))

print("---------------------------------------------")

print(np.imag(complex\_signal))

print("---------------------------------------------")

plt.figure()

plt.title("OFDM Symbol")

plt.plot(complex\_signal.real)

plt.plot(complex\_signal.imag)

plt.show()

plt.figure()

plt.title("OFDM complex spectrum")

plt.xlabel("Normalised frequencies")

plt.ylabel("Frequency amplitudes")

plt.plot(

    np.abs(np.fft.fft(complex\_signal[-totalFreqSamples:]) / totalFreqSamples))

plt.show()

*# read image*

img = Image.open("DC4\_300x200.pgm")

tx\_byte = np.array(img).ravel()  *# one dimension array*

arr = np.asarray(img)

plt.imshow(arr, cmap='gray')

plt.show()

*# append dummy bytes in order to make the data array is a whole multiple of nbytes*

pad\_num = nbytes - tx\_byte.shape[0] % nbytes

tx\_byte = np.pad(tx\_byte, (0, pad\_num), mode="constant", constant\_values=127)

*# OFDM encoding*

complex\_signal = np.array([ofdm.encode(tx\_byte[i:i+nbytes])

                           for i in range(0, tx\_byte.size, nbytes)]).ravel()

*# modulate*

base\_signal = pyofdm.nyquistmodem.mod(complex\_signal)

*# add some random length dummy zero to the start of the signal*

random\_extra\_zero = 50

base\_signal = np.pad(base\_signal, (random\_extra\_zero, 0), mode="constant")

*# save it as a wav file*

wav.write("ofdm44100.wav", 44100, base\_signal)

## 1.3

from PIL import Image

import numpy as np

import scipy.io.wavfile as wav

import pyofdm.codec

import pyofdm.nyquistmodem

import matplotlib.pyplot as plt

*# Number of total frequency samples*

totalFreqSamples = 2048

*# Number of useful data carriers / frequency samples*

sym\_slots = 1512

*# QAM Order*

QAMorder = 2

*# Total number of bytes per OFDM symbol*

nbytes = sym\_slots \* QAMorder // 8

*# Distance of the evenly spaced pilots*

distanceOfPilots = 12

pilotlist = pyofdm.codec.setpilotindex(nbytes, QAMorder, distanceOfPilots)

ofdm = pyofdm.codec.OFDM(pilotAmplitude=16/9,

                         nData=nbytes,

                         pilotIndices=pilotlist,

                         mQAM=QAMorder,

                         nFreqSamples=totalFreqSamples)

samp\_rate, base\_signal = wav.read("ofdm44100.wav")

*# add zeros to the base\_signal*

extra\_zero\_length = 60

base\_signal = np.pad(base\_signal, (0, extra\_zero\_length), "constant")

complex\_signal = pyofdm.nyquistmodem.demod(base\_signal)

*# find the start of the OFDM symbol*

searchRangeForPilotPeak = 8

cc, sumofimag, offset = ofdm.findSymbolStartIndex(

    complex\_signal, searchrangefine=searchRangeForPilotPeak)

print("Symbol start sample index =", offset)

plt.plot(cc)

plt.title("cc")

plt.show()

search\_range = np.arange(-searchRangeForPilotPeak, searchRangeForPilotPeak)

plt.plot(search\_range, sumofimag)

plt.title("sumofimag")

plt.show()

Nsig\_sym = 159

ofdm.initDecode(complex\_signal, 25)

rx\_byte = np.uint8([ofdm.decode()[0] for i in range(Nsig\_sym)]).ravel()

rx\_byte = 255 - rx\_byte

rx\_byte = rx\_byte[:60000].reshape(200, 300)

receive\_img = Image.fromarray(rx\_byte)

plt.imshow(receive\_img, plt.cm.gray)

plt.show()

*# bit error ratio*

origin\_img = Image.open("DC4\_300x200.pgm")

origin\_img = np.array(origin\_img)

ber = np.sum(origin\_img != receive\_img) / origin\_img.size

print("Bit error ratio = ", ber)

## 1.4

from PIL import Image

import numpy as np

import scipy.io.wavfile as wav

import pyofdm.codec

import pyofdm.nyquistmodem

import matplotlib.pyplot as plt

*# Number of total frequency samples*

totalFreqSamples = 2048

*# Number of useful data carriers / frequency samples*

sym\_slots = 1512

*# QAM Order*

QAMorder = 2

*# Total number of bytes per OFDM symbol*

nbytes = sym\_slots \* QAMorder // 8

*# Distance of the evenly spaced pilots*

distanceOfPilots = 12

pilotlist = pyofdm.codec.setpilotindex(nbytes, QAMorder, distanceOfPilots)

ofdm = pyofdm.codec.OFDM(pilotAmplitude=16/9,

                         nData=nbytes,

                         pilotIndices=pilotlist,

                         mQAM=QAMorder,

                         nFreqSamples=totalFreqSamples)

**def** receive(wave\_file):

    samp\_rate, base\_signal = wav.read(wave\_file)

*# append some extra zeros to the base\_signal*

    extra\_pad\_length = 60

    base\_signal = np.pad(base\_signal, (0, extra\_pad\_length), "constant")

    complex\_signal = pyofdm.nyquistmodem.demod(base\_signal)

    Nsig\_sym = 159

    ofdm.initDecode(complex\_signal, 25)

    rx\_byte = np.uint8([ofdm.decode()[0] for i in range(Nsig\_sym)]).ravel()

    rx\_byte = 255 - rx\_byte

    rx\_byte = rx\_byte[:60000].reshape(200, 300)

    receive\_img = Image.fromarray(rx\_byte)

    plt.imshow(receive\_img, plt.cm.gray)

    plt.show()

*# calculate bit error ratio*

    origin\_img = Image.open("DC4\_300x200.pgm")

    origin\_img = np.array(origin\_img)

    ber = np.sum(origin\_img != receive\_img) / origin\_img.size

    print("Bit error ratio = ", ber)

*# receive("ofdm44100\_reverb.wav")*

*# receive("ofdm44100\_reverb\_B20.wav")*

*# receive("ofdm44100\_reverb\_B80.wav")*

*# receive("ofdm44100\_reverb\_M20.wav")*

*# receive("ofdm44100\_reverb\_M80.wav")*

*# receive("ofdm44100\_white\_noise0.8.wav")*

receive("ofdm44100\_white\_noise0.05.wav")

receive("ofdm44100\_white\_noise0.015.wav")

## 1.5

from PIL import Image

import numpy as np

import scipy.io.wavfile as wav

import pyofdm.codec

import pyofdm.nyquistmodem

import matplotlib.pyplot as plt

from reedsolo import RSCodec

from reedsolo import ReedSolomonError

N, K = 255, 223

rsc = RSCodec(N-K, nsize=N)

tx\_im = Image.open("DC4\_300x200.pgm")

tx\_byte = np.append(np.array(tx\_im, dtype="uint8").flatten(),

                    np.zeros(K-tx\_im.size[1]\*tx\_im.size[0] % K, dtype="uint8"))

tx\_enc = np.empty(0, "uint8")

for i in range(0, tx\_im.size[1]\*tx\_im.size[0], K):

    tx\_enc = np.append(tx\_enc, np.uint8(rsc.encode(tx\_byte[i:i+K])))

*# Number of total frequency samples*

totalFreqSamples = 2048

*# Number of useful data carriers / frequency samples*

sym\_slots = 1512

*# QAM Order*

QAMorder = 2

*# Total number of bytes per OFDM symbol*

nbytes = sym\_slots \* QAMorder // 8

*# Distance of the evenly spaced pilots*

distanceOfPilots = 12

pilotlist = pyofdm.codec.setpilotindex(nbytes, QAMorder, distanceOfPilots)

ofdm = pyofdm.codec.OFDM(pilotAmplitude=16/9,

                         nData=nbytes,

                         pilotIndices=pilotlist,

                         mQAM=QAMorder,

                         nFreqSamples=totalFreqSamples)

*# append dummy bytes in order to make the data array is a whole multiple of nbytes*

pad\_num = nbytes - tx\_enc.shape[0] % nbytes

tx\_enc = np.pad(tx\_enc, (0, pad\_num), mode="constant", constant\_values=127)

*# OFDM encoding*

complex\_signal = np.array([ofdm.encode(tx\_enc[i:i+nbytes])

                           for i in range(0, tx\_enc.size, nbytes)]).ravel()

*# modulate*

base\_signal = pyofdm.nyquistmodem.mod(complex\_signal)

*# add some random length dummy zero to the start of the signal*

random\_pad\_length = 50

base\_signal = np.pad(base\_signal, (random\_pad\_length, 0), mode="constant")

*# save it as a wav file*

wav.write("ofdm44100\_channel.wav", 44100, base\_signal)

samp\_rate, base\_signal = wav.read("ofdm44100\_channel.wav")

*# append some extra zeros to the base\_signal*

extra\_pad\_length = 60

base\_signal = np.pad(base\_signal, (0, extra\_pad\_length), "constant")

complex\_signal = pyofdm.nyquistmodem.demod(base\_signal)

*# find the start of the OFDM symbol*

searchRangeForPilotPeak = 8

cc, sumofimag, offset = ofdm.findSymbolStartIndex(

    complex\_signal, searchrangefine=searchRangeForPilotPeak)

print("Symbol start sample index =", offset)

Nsig\_sym = 183

ofdm.initDecode(complex\_signal, 25)

rx\_enc = np.uint8([ofdm.decode()[0] for i in range(Nsig\_sym)]).ravel()

rx\_enc = 255 - rx\_enc

rx\_byte = np.empty(0, dtype="uint8")

for i in range(0, tx\_im.size[1]\*tx\_im.size[0]\*N//K, N):

    try:

        rx\_byte = np.append(rx\_byte, np.uint8(rsc.decode(rx\_enc[i:i+N])[0]))

    except ReedSolomonError:

        rx\_byte = np.append(rx\_byte, rx\_enc[i:i+K])

rx\_byte = rx\_byte[:60000].reshape(200, 300)

receive\_img = Image.fromarray(rx\_byte)

plt.imshow(receive\_img, plt.cm.gray)

plt.show()

*# calculate bit error ratio*

origin\_img = Image.open("DC4\_300x200.pgm")

origin\_img = np.array(origin\_img)

ber = np.sum(origin\_img != receive\_img) / origin\_img.size

print("Bit error ratio = ", ber)

N, K = 255, 223

rsc = RSCodec(N-K, nsize=N)

tx\_im = Image.open("DC4\_300x200.pgm")

tx\_byte = np.append(np.array(tx\_im, dtype="uint8").flatten(),

                    np.zeros(K-tx\_im.size[1]\*tx\_im.size[0] % K, dtype="uint8"))

tx\_enc = np.empty(0, "uint8")

for i in range(0, tx\_im.size[1]\*tx\_im.size[0], K):

    tx\_enc = np.append(tx\_enc, np.uint8(rsc.encode(tx\_byte[i:i+K])))

*# Number of total frequency samples*

totalFreqSamples = 2048

*# Number of useful data carriers / frequency samples*

sym\_slots = 1512

*# QAM Order*

QAMorder = 2

*# Total number of bytes per OFDM symbol*

nbytes = sym\_slots \* QAMorder // 8

*# Distance of the evenly spaced pilots*

distanceOfPilots = 12

pilotlist = pyofdm.codec.setpilotindex(nbytes, QAMorder, distanceOfPilots)

ofdm = pyofdm.codec.OFDM(pilotAmplitude=16/9,

                         nData=nbytes,

                         pilotIndices=pilotlist,

                         mQAM=QAMorder,

                         nFreqSamples=totalFreqSamples)

*# append dummy bytes in order to make the data array is a whole multiple of nbytes*

pad\_num = nbytes - tx\_enc.shape[0] % nbytes

tx\_enc = np.pad(tx\_enc, (0, pad\_num), mode="constant", constant\_values=127)

*# OFDM encoding*

complex\_signal = np.array([ofdm.encode(tx\_enc[i:i+nbytes])

                           for i in range(0, tx\_enc.size, nbytes)]).ravel()

*# modulate*

base\_signal = pyofdm.nyquistmodem.mod(complex\_signal)

*# add some random length dummy zero to the start of the signal*

random\_pad\_length = 50

base\_signal = np.pad(base\_signal, (random\_pad\_length, 0), mode="constant")

*# save it as a wav file*

*# wav.write("ofdm44100\_channel.wav", 44100, base\_signal)*

*# receive("ofdm44100\_white\_noise0.05.wav")*

*# receive("ofdm44100\_white\_noise0.015.wav")*

samp\_rate, base\_signal = wav.read("ofdm44100\_channel.wav")

*# append some extra zeros to the base\_signal*

extra\_pad\_length = 60

base\_signal = np.pad(base\_signal, (0, extra\_pad\_length), "constant")

complex\_signal = pyofdm.nyquistmodem.demod(base\_signal)

*# find the start of the OFDM symbol*

searchRangeForPilotPeak = 8

cc, sumofimag, offset = ofdm.findSymbolStartIndex(

    complex\_signal, searchrangefine=searchRangeForPilotPeak)

print("Symbol start sample index =", offset)

Nsig\_sym = 183

ofdm.initDecode(complex\_signal, 25)

rx\_enc = np.uint8([ofdm.decode()[0] for i in range(Nsig\_sym)]).ravel()

rx\_enc = 255 - rx\_enc

rx\_byte = np.empty(0, dtype="uint8")

for i in range(0, tx\_im.size[1]\*tx\_im.size[0]\*N//K, N):

    try:

        rx\_byte = np.append(rx\_byte, np.uint8(rsc.decode(rx\_enc[i:i+N])[0]))

    except ReedSolomonError:

        rx\_byte = np.append(rx\_byte, rx\_enc[i:i+K])

rx\_byte = rx\_byte[:60000].reshape(200, 300)

receive\_img = Image.fromarray(rx\_byte)

plt.imshow(receive\_img, plt.cm.gray)

*# calculate bit error ratio*

origin\_img = Image.open("DC4\_300x200.pgm")

origin\_img = np.array(origin\_img)

ber = np.sum(origin\_img != receive\_img) / origin\_img.size

print("Bit error ratio = ", ber)