****

ENG4052: Digital Communication 4 (2022-23)

Lab4: Forward Error Correction

**Haoshi Huang (2635088H)**

**Submission Date**

**08/03/2023**

**1 Introduction**

This lab will use two kinds of **Forward Error Correction (FEC)**, including **BCH Code (BCH) and Convolutional Code (Convn)**, to overcome the effects of AWGN in the communication channel. This time we only use QPSK modulation. Firstly, we will plot the BER vs SNR using BCH and Convn respectively. Secondly, we will set SNR to 3 dB using BCH(7, 4), BCH(15, 5), BCH(31, 6) and BCH(63, 10) and compare these code rate. Finally, we will set SNR to 0 dB using both BCH as the outer code and Convn as the inner code comparing with the previous results to get the lowest BER under what conditions.

We will use Library komm 0.7.1 to modulate and demodulate signal and create AWGN. Lib komm also provide BCH and Convn coder/decoder methods. Library NumPy1.23, Scipy1.9.2, matplotlib3.6, and Pillow9.2.0 are also imported to source files to implement the lab target.

**2 Bose-Chaudhuri-Hocquenghem Codes (BCH)**

**2.1 Using BCH**

In theory, according to the given monitor code length and maximum number of error correction bits, which can obtain corresponding generating polynomial. Then we can use generating polynomial to implementing encoding and decoding.

In code, I continued to use the **Class imgInfo** and **Class modConfig**. Since multiple monitoring code techniques are required this time, additionally method *encodeFEC* and *decodeFEC* are defined, which are both extracted from method *repeatTransmit*. I use python method *isinstance* is to judgment the current FEC technique when a real parameter is passed in these two methods as shown in the Fig. 2.1 and Fig. 2.2.

图形用户界面, 文本, 应用程序

描述已自动生成

*Figure 2.1 method encodeFEC*

图形用户界面, 文本, 应用程序

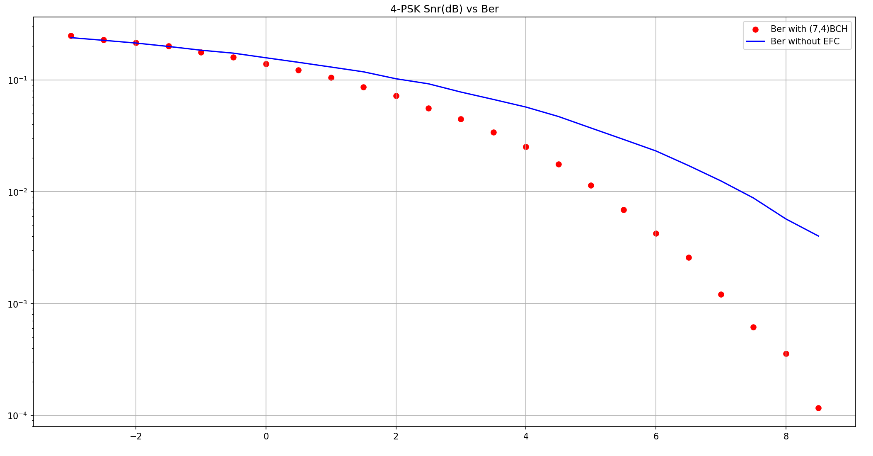
描述已自动生成

*Figure 2.2 method decodeFEC*

When using *BCHCode* of lib komm, it’s **worth mentioning** that we need to split the binary data array of image by the BCH message code length, equaling to *BCHCoder.length* attributes.

**2.2 Under different SNR**

With QPSK modulation, we plot the Ber vs Snr under different SNR from -3 to 9 dB using BCH(7, 4), BCH(15, 5), BCH(31, 6) and BCH(63, 10) as shown in the Fig. 2.3, Fig. 2.4, Fig. 2.5, & Fig. 2.6. Taking BCH(7, 4) means every 4 bits are encoded, increasing by 3 bits and finally get 7 bits. In the figs, the blue line means Ber without correction code, and the red scattered points means Ber with BCH code.

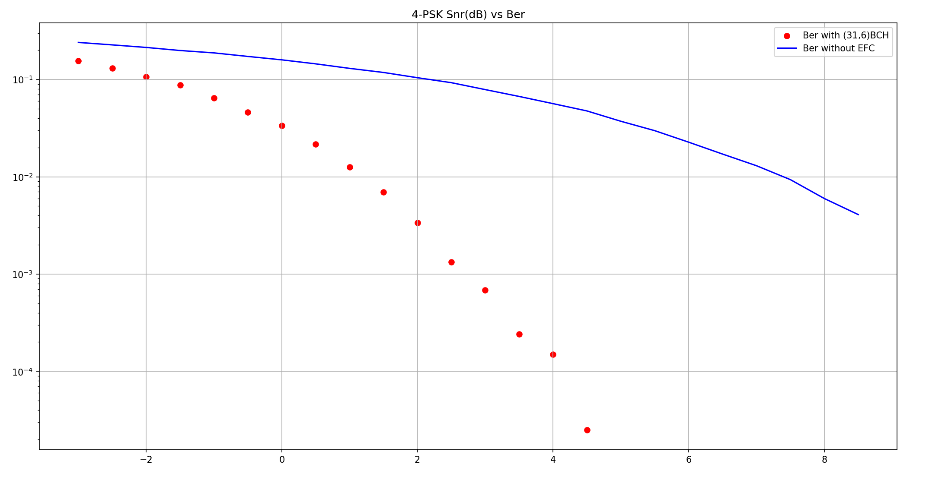


*Figure 2.3 BCH(7, 4)*

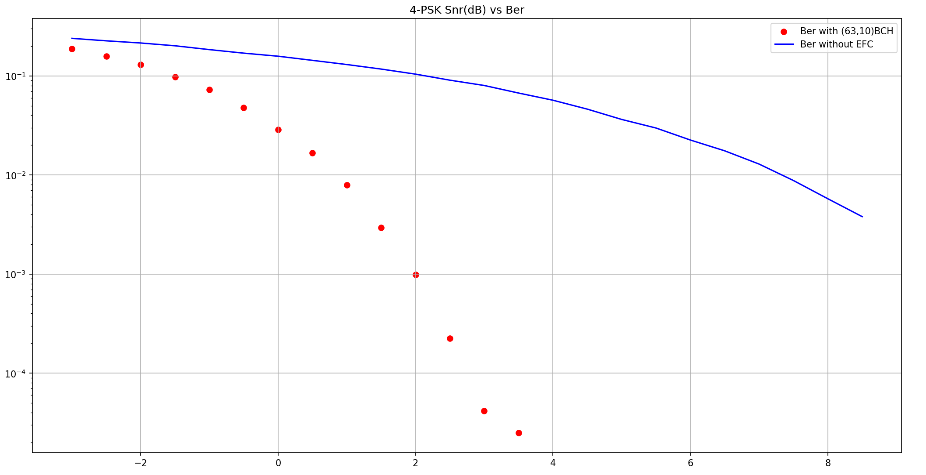
图表, 散点图

描述已自动生成

*Figure 2.4 BCH(15, 5)*



*Figure 2.5 BCH(31, 6)*

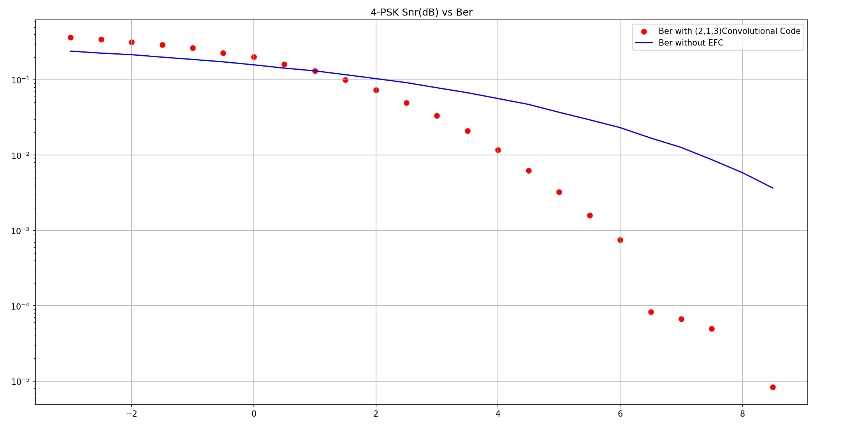


*Figure 2.6 BCH(63, 10)*

**3 Convolutional Codes (Convn)**

**3.1 Using Convn**

Compared to BCH codes, the structures of Convn encoders are possible more complex. The Convn takes into account the previous input bits as the current output bits. Additionally, output bits are generated from multiple modulo-two adders. In the test code, I only use the code only test [[0o7, 0o5]] as generating polynomial, which will create a Convn(2, 1, 3) meaning 2 output bits, 1 input bit and 3 overall constraint length including current input bit. I also plot Ber vs Snr from -3 dB to 9 dB using Convn(2, 1, 3) as shown in the Fig. 3.1.



*Figure 3.1 Convn(2, 1, 3)*

Using *ConvolutionalCode* of lib komm different from BCHCode, we **do not** need to split the binary array of image. But we need to fill zeros when decoding as shown in the Fig. 3.2. The operation of filling zeros can get the right image in each pixel. And we need to discard the first *tblen* bits. Parameters tblen always 5 or 6 times than constraint length.

图形用户界面, 文本

中度可信度描述已自动生成

*Figure 3.2 Convn decoder needs filling zeros*

**4 Conclusion**

**4.1 About BCH Codes**

Based on code running results in the Fig. 2.3, Fig. 2.4, Fig. 2.5 & Fig. 2.6 and running time, at the same snr, the higher order of BCH codes, the capacity of monitoring and correction of errors is stronger, which means the ber of BCH(63, 10) is lower than BCH(31, 6), similar to the relation among BCH(31, 6), BCH(15, 5) and BCH(7,4). But from the point of view of code rate, the higher order of BCH code, the code rate is lower, which are (10/63) < (6/31) < (5/15) < (4/7).

**4.2 About Concatenated Codes**

We use BCH code as outer coder and Convn as inner coder. We set SNR to 3 dB (as high SNR) and 0 dB (as low SNR) with only BCH, only Convn and both. The result of BER are as follows: when snr = 3.0 dB, BCH(4.42%), Convn(3.38%), Conca(3.18%); when snr = 0 dB, BCH(14.1%), Con(19.87%), Conca(19.12%). The conclusion is that when the higher SNR, BCH is worse than Convn; when the lower SNR, BCH is better than Convn. The BER of concatenated codes is always between previous two and it approaches the result of only Convn codes, which means Convn codes has strong impact.

**Appendix:**

**ImgInfo.py**

'''

Author : Eureke

Date : 2023-03-06 14:29:36

LastEditors : Marcus Wong

LastEditTime : 2023-03-08 17:33:57

Description :

'''

import numpy as np

from PIL import Image

from matplotlib import pyplot as plt

class imgInfo:

def \_\_init\_\_(self, fp, word\_len):

self.imBin, self.imSize = self.openImagetoBin(fp)

self.Npixels = self.imSize[1] \* self.imSize[0]

self.word\_len = word\_len

self.imBin\_encoded = None

self.rx\_bin = None

# open image file

def openImagetoBin(self, filePath):

im = Image.open(filePath)

if (True):

plt.figure()

plt.imshow(np.array(im),cmap="gray",vmin=0,vmax=255)

plt.show()

imBin = np.unpackbits(np.array(im))

print('original shape: ', imBin.shape)

return imBin, im.size

# show demodulated image

def displayDemodImage(self):

# demod signal with noise

rx\_im = np.packbits(self.rx\_bin).reshape(self.imSize[1], self.imSize[0])

plt.figure()

plt.imshow(np.array(rx\_im),cmap="gray",vmin=0,vmax=255)

plt.show()

**ModConfig.py**

'''

Author : Eureke

Date : 2023-03-08 14:54:30

LastEditors : Marcus Wong

LastEditTime : 2023-03-08 17:35:04

Description :

'''

import komm

class modConfig:

def \_\_init\_\_(self, method, orders, snr, base\_amplitudes, phase\_offset):

self.method = method

self.orders = orders

self.snr = snr

self.base\_amplitudes = base\_amplitudes

self.phase\_offset = phase\_offset

self.modulation, self.awgn = self.set\_modulation()

# create komm's modulation object

def set\_modulation(self):

if self.method == 'psk':

modulation = komm.PSKModulation(self.orders, amplitude=self.base\_amplitudes, phase\_offset=self.phase\_offset)

elif self.method == 'qam':

modulation = komm.QAModulation(self.orders, base\_amplitudes=self.base\_amplitudes, phase\_offset=self.phase\_offset)

# Additive white gaussian noise(AWGN)

awgn = komm.AWGNChannel(self.snr)

return modulation, awgn

# self-add snr

def set\_snr(self, new\_snr):

self.snr = new\_snr

self.modulation, self.awgn = self.set\_modulation()

**SimTrans.py**

'''

Author : Eureke

Date : 2023-03-08 15:06:27

LastEditors : Marcus Wong

LastEditTime : 2023-03-08 20:47:30

Description :

'''

import numpy as np

from matplotlib import pyplot as plt

import komm

from ModConfig import modConfig

# input imag and FEC coder to encode

def encodeFEC(img, coder):

if (isinstance(coder, komm.\_error\_control\_block.BCHCode)):

BCHCoder = coder

coder\_type = "BCH"

# there is a potential bug about (img.imBin.size/BCHCoder.dimension) if at the last code is not enough BCHCoder.dimension need to fill zero

imBin\_copy = np.copy(img.imBin.reshape(int(img.imBin.size/BCHCoder.dimension), BCHCoder.dimension))

# print('The shape after grouping: ', imBin\_copy.shape)

img.imBin\_encoded = np.array([BCHCoder.encode(i) for i in imBin\_copy]).ravel()

# print("The shape after BCH code: ", img.imBin\_encoded.shape)

elif (isinstance(coder, komm.\_error\_control\_convolutional.ConvolutionalCode)):

ConvnCoder = coder

coder\_type = "Convn"

# create Convn encoder

encoder = komm.ConvolutionalStreamEncoder(ConvnCoder, initial\_state=0)

imBin\_copy = np.copy(img.imBin)

img.imBin\_encoded = encoder(imBin\_copy)

return coder\_type, img.imBin\_encoded

def decodeFEC(rx\_demod, coder):

if (isinstance(coder, komm.\_error\_control\_block.BCHCode)):

BCHCoder = coder

coder\_type = "BCH"

# BCH code check and error recovery

rx\_demod = rx\_demod.reshape(int(rx\_demod.size/BCHCoder.length), BCHCoder.length)

rx\_bin = np.array([BCHCoder.decode(i) for i in rx\_demod]).ravel()

elif (isinstance(coder, komm.\_error\_control\_convolutional.ConvolutionalCode)):

ConvnCoder = coder

coder\_type = "Convn"

tblen = 18

decoder = komm.ConvolutionalStreamDecoder(ConvnCoder, traceback\_length=tblen, input\_type="hard")

# print(rx\_demod.shape)

# print(np.zeros(2\*tblen, dtype=np.int32).shape)

# print(type(rx\_demod[0]))

decoded\_middle = decoder(np.append(rx\_demod, np.zeros(2\*tblen, dtype=np.int32)))

rx\_bin = decoded\_middle[tblen:]

return coder\_type, rx\_bin.astype(np.bool\_)

# stimulate transmit single img with correction

def transmission(img, mod\_config, coder):

# transmission with FEC correction

# modulated signal

tx\_data = mod\_config.modulation.modulate(img.imBin\_encoded)

# add awgn

rx\_data = mod\_config.awgn(tx\_data)

# demodulate at receiver

rx\_demod = mod\_config.modulation.demodulate(rx\_data)

# decode using FEC decoder

coder\_type, img.rx\_bin = decodeFEC(rx\_demod, coder)

# compute ber with FEC

ber = practiceBer(img.imBin, img.rx\_bin)

print('bit error ratio with {} code: {:.3}%'.format(coder\_type, ber \* 100))

if (False):

img.displayDemodImage()

return ber

# stimulate transmit single img without correction

def transmissionNoCorrection(img, mod\_config):

# transmission with no correction

tx\_data = mod\_config.modulation.modulate(img.imBin)

rx\_data = mod\_config.awgn(tx\_data)

rx\_bin = mod\_config.modulation.demodulate(rx\_data)

ber = practiceBer(img.imBin, rx\_bin)

print('bit error ratio without FEC code: {:.3}%'.format(ber \* 100))

if (False):

img.displayDemodImage()

return ber

def repeatTransmit(img, coder, method, orders, snr\_ctrl, base\_amplitudes=1., phase\_offset=0.):

print("Start " + str(orders) + '-' + method + "modulation:")

# use FEC to encode img

coder\_type, \_ = encodeFEC(img, coder)

# initial modulation config

# snr from -3 to 9 dB

mod\_config = modConfig(method, orders, snr\_ctrl[0], base\_amplitudes, phase\_offset)

# save ber and snr of each trasmission single image

correction\_ber\_out = np.empty(0)

nocorrection\_ber\_out = np.empty(0)

snr\_out = np.empty(0)

for i in np.arange(snr\_ctrl[0], snr\_ctrl[1], snr\_ctrl[2]):

snr = 10\*\*(i/10.)

mod\_config.set\_snr(snr)

correction\_ber = transmission(img, mod\_config, coder)

nocorrection\_ber = transmissionNoCorrection(img, mod\_config)

correction\_ber\_out = np.append(correction\_ber\_out, correction\_ber)

nocorrection\_ber\_out = np.append(nocorrection\_ber\_out, nocorrection\_ber)

snr\_out = np.append(snr\_out, i)

print('snr(dB): ', i)

# print('snr: ', mod\_config.snr)

print("Ber with correction: ", correction\_ber\_out)

print("Ber without correction: ", nocorrection\_ber\_out)

print("SNR: ", snr\_out)

if (True):

plt.figure()

plt.title(str(orders) + '-' + method.upper() + ' Snr(dB) vs Ber')

if (coder\_type == "BCH"):

BCHCoder = coder

plt.scatter(snr\_out, correction\_ber\_out, color='r', label=('Ber with ' + '(' + str(BCHCoder.length) + ',' + str(BCHCoder.dimension) +')BCH'))

elif (coder\_type == "Convn"):

ConvnCoder = coder

plt.scatter(snr\_out, correction\_ber\_out, color='r', label=('Ber with ' + '(' + str(ConvnCoder.num\_output\_bits) + ',' + str(ConvnCoder.num\_input\_bits) + ',' + str(ConvnCoder.overall\_constraint\_length + 1) +')Convolutional Code'))

plt.plot(snr\_out, nocorrection\_ber\_out, color='b', label='Ber without EFC')

plt.yscale("log")

plt.grid(True)

plt.legend()

plt.show()

# compute ber in practice

practiceBer = lambda tx\_bin, rx\_bin : np.sum([pix[0] != pix[1] for pix in zip(tx\_bin, rx\_bin)]) / tx\_bin.size

**BCHCodes.py**

'''

Author : Eureke

Date : 2023-03-05 09:22:37

LastEditors : Marcus Wong

LastEditTime : 2023-03-08 21:19:49

Description :

'''

import numpy as np

import komm

from ImgInfo import imgInfo

from ModConfig import modConfig

from SimTrans import repeatTransmit

if \_\_name\_\_ == "\_\_main\_\_":

# open image and binary information

fp = './Lab4/DC4\_150x100.pgm'

# fp = './Lab3/DC4\_640x480.pgm'

word\_len = 8 # 256 bits per pixel

img = imgInfo(fp, word\_len)

# BCH code

# Length = 2^miu - 1

# message length = tau = 1

# code = komm.BCHCode(mu=3, tau=1)

# n, k = code.length, code.dimension

# print(code.generator\_polynomial)

# print(code.generator\_matrix)

# message = np.array([1, 0, 0, 1])

# recvword = code.encode(message)

# print(recvword)

# message\_decoded = code.decode(recvword)

# print(message\_decoded)

snr\_ctrl = [-3., 9., 0.5]

snr\_ctrl = [3., 3.5, 0.5]

snr\_ctrl = [0., 0.5, 0.5]

# qpsk modulation with BCH code

repeatTransmit(img=img, coder=komm.BCHCode(mu=3, tau=1), method='psk', orders=4, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, coder=komm.BCHCode(mu=4, tau=3), method='psk', orders=4, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, coder=komm.BCHCode(mu=5, tau=7), method='psk', orders=4, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, coder=komm.BCHCode(mu=6, tau=13), method='psk', orders=4, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

**ConvnCodes.py**

'''

Author : Eureke

Date : 2023-03-08 16:30:21

LastEditors : Marcus Wong

LastEditTime : 2023-03-08 21:11:47

Description :

'''

import numpy as np

import komm

from ImgInfo import imgInfo

from ModConfig import modConfig

from SimTrans import repeatTransmit

if \_\_name\_\_ == '\_\_main\_\_':

# open image and binary information

fp = './Lab4/DC4\_150x100.pgm'

# fp = './Lab3/DC4\_640x480.pgm'

word\_len = 8 # 256 bits per pixel

img = imgInfo(fp, word\_len)

'''

print(img.imBin[:16])

code = komm.ConvolutionalCode(feedforward\_polynomials=[[0o7, 0o5]])

encoder = komm.ConvolutionalStreamEncoder(code, initial\_state=0)

new\_m = encoder(img.imBin)

print(new\_m[:32])

decoder = komm.ConvolutionalStreamDecoder(code, traceback\_length=4, input\_type="hard")

decoded\_m\_final = decoder(np.append(new\_m[:32], np.zeros(8, dtype=np.int32)))

# decoded\_m\_final = decoder(np.zeros(2\*8, dtype=np.int32))

print(decoded\_m\_final[4:])

print(decoded\_m\_final.shape)

'''

snr\_ctrl = [-3., 9., 0.5]

snr\_ctrl = [3., 3.5, 0.5]

snr\_ctrl = [0., 0.5, 0.5]

# qpsk modulation with convn code

repeatTransmit(img=img, coder=komm.ConvolutionalCode(feedforward\_polynomials=[[0o7, 0o5]]), method='psk', orders=4, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

**ConcatenatedCodes.py**

'''

Author : Eureke

Date : 2023-03-08 20:28:50

LastEditors : Marcus Wong

LastEditTime : 2023-03-08 21:16:57

Description :

'''

import numpy as np

import komm

from ImgInfo import imgInfo

from ModConfig import modConfig

from SimTrans import practiceBer

def concatenatedTransmit(img, inner\_coder, outer\_coder, method, orders, snr, base\_amplitudes=1., phase\_offset=0.):

print("Start " + str(orders) + '-' + method + "modulation:")

# inner&outer FEC encode

BCHCoder = outer\_coder

imBin\_copy = np.copy(img.imBin.reshape(int(img.imBin.size/BCHCoder.dimension), BCHCoder.dimension))

# print('The shape after grouping: ', imBin\_copy.shape)

img.imBin\_encoded = np.array([BCHCoder.encode(i) for i in imBin\_copy]).ravel()

ConvnCoder = inner\_coder

# create Convn encoder

encoder = komm.ConvolutionalStreamEncoder(ConvnCoder, initial\_state=0)

imBin\_copy = np.copy(img.imBin\_encoded)

img.imBin\_encoded = encoder(imBin\_copy)

# initial modulation config

mod\_config = modConfig(method, orders, snr, base\_amplitudes, phase\_offset)

mod\_config.set\_snr(10\*\*(snr/10.))

# modulated signal

tx\_data = mod\_config.modulation.modulate(img.imBin\_encoded)

# add awgn

rx\_data = mod\_config.awgn(tx\_data)

# demodulate at receiver

rx\_demod = mod\_config.modulation.demodulate(rx\_data)

# decode demod signal

tblen = 18

decoder = komm.ConvolutionalStreamDecoder(ConvnCoder, traceback\_length=tblen, input\_type="hard")

decoded\_middle = decoder(np.append(rx\_demod, np.zeros(2\*tblen, dtype=np.int32)))

rx\_bin\_inner = decoded\_middle[tblen:]

rx\_bin\_inner = rx\_bin\_inner.reshape(int(rx\_bin\_inner.size/BCHCoder.length), BCHCoder.length)

rx\_bin = np.array([BCHCoder.decode(i) for i in rx\_bin\_inner]).ravel()

img.rx\_bin = rx\_bin

ber = practiceBer(img.imBin, img.rx\_bin)

print('ber: ', ber)

print('bit error ratio with BCH & Convn code: {:.3}%'.format(ber \* 100))

if (True):

img.displayDemodImage()

if \_\_name\_\_ == '\_\_main\_\_':

# open image and binary information

fp = './Lab4/DC4\_150x100.pgm'

# fp = './Lab3/DC4\_640x480.pgm'

word\_len = 8 # 256 bits per pixel

img = imgInfo(fp, word\_len)

concatenatedTransmit(img=img, inner\_coder=komm.ConvolutionalCode(feedforward\_polynomials=[[0o7, 0o5]]), outer\_coder=komm.BCHCode(mu=3, tau=1), method='psk', orders=4, snr=3., base\_amplitudes=1., phase\_offset=0.)

concatenatedTransmit(img=img, inner\_coder=komm.ConvolutionalCode(feedforward\_polynomials=[[0o7, 0o5]]), outer\_coder=komm.BCHCode(mu=3, tau=1), method='psk', orders=4, snr=0., base\_amplitudes=1., phase\_offset=0.)