University of Glasgow

ENG4052: Digital Communication 4 (2022-23)

Lab3: Bit Errors and Parity Checking

Haoshi Huang (2635088H)

Submission Date

22/2/2023

1 Introduction

Firstly, this lab will simulate transmission process of BPSK and QPSK in the communication channel with **Additive White Gaussian Noise (AWGN)** at different **Signal Noise Ratio (snr)**. Secondly, we will use a simple Parity Checking to mitigate the effects of AWGN in PSK and QAM with **Bit-error-ratio (ber)** to evaluate the effects. Finally, we plot the ber as a function of snr in dB respectively, in order to draw conclusion.

We will use Library komm 0.7.1 to modulate and demodulate signal and create AWGN. Library NumPy1.23 is imported to implement advanced math operation. Library Scipy 1.9.2 has *special.erfc(snr)* function to compute theoretical ber. Library Pillow 9.2.0 is used to read image file. Library matplotlib 3.6 is used to display graphics and plot the bit error ratio as a function of signal noise ratio in dB.

2 Noisy Channel Simulation

2.1 Digital image data

Taking a grayscale image as an example as shown in the Fig. 2.1, it's read as 2-dimensional arrays with the help of Lib pillow. Using the function *numpy.unpackbits()* changes the data to single dimension, which is helpful for processing data and transmission.

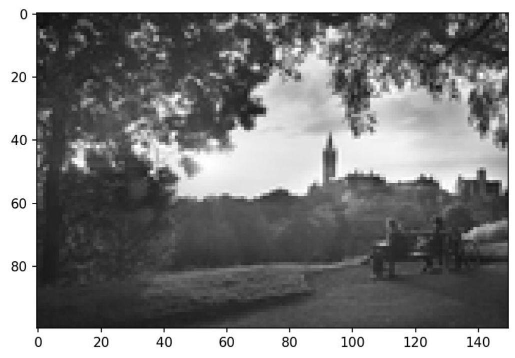
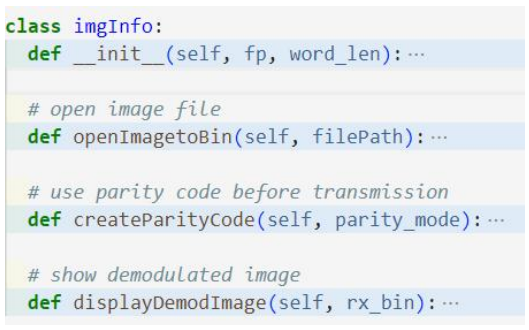


Figure 2.1 a grayscale image

For reusing code, **Class imgInfo** is defined as shown in the Fig. 2.2, which has member property including size of image, number of bits per pixel, etc.



```
class imgInfo:
    def __init__(self, fp, word_len):···

    # open image file
    def openImagetoBin(self, filePath):···

    # use parity code before transmission
    def createParityCode(self, parity_mode):···

    # show demodulated image
    def displayDemodImage(self, rx_bin):···
```

Figure 2.2 Class imgInfo

Method *openImagetoBin* is used to open and show image file and convert the image data to binary signal.

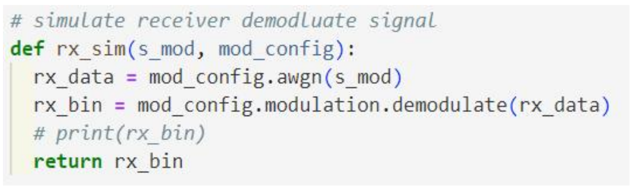
Method *createParityCod*e (referring to **3 Parity Check**) can generate binary code with parity information according to the parameter *parity_mode* (0 means even checking, 1 means odd checking).

Method *displayDemodImage* is used to show demodulated image, comparing with the original image.

2.2 Additive White Gaussian Noise

Additive White Gaussian Noise (AWGN) is a type of basic random noise, which is always used to simulate the effects of practical noise in electronic system, such as communication system. ‘Additive’ noise means that the relationship between the noise and signal is addition. ‘White’ noise means the power spectrum density of stochastic noise is a constant. ‘Gaussian’ noise means that the stochastic noise obeys the Gaussian Distribution.

In the code, AWGN is hypothetically added at the moment of receiving single byte modulated signal.



```
# simulate receiver demodluate signal
def rx_sim(s_mod, mod_config):
  rx_data = mod_config.awgn(s_mod)
  rx_bin = mod_config.modulation.demodulate(rx_data)
  # print(rx_bin)
  return rx_bin
```

Figure 2.3 Simulate receiver

Taking QPSK as an example, the Fig. 2.4 plot the modulated signal with stochastic noise with parameters *phase_offset = np.pi /2* (the default is *np.pi/4*) in the condition of *SNR* from *2 dB* to *9 dB*. The higher SNR is, the more concentrated the signal is.

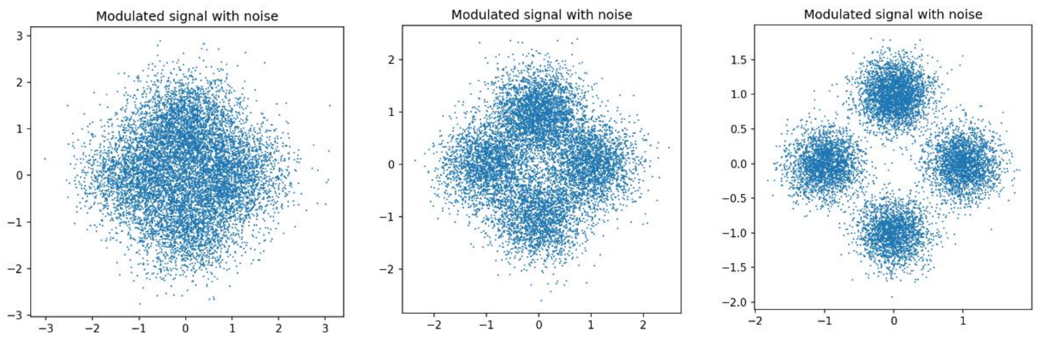


Figure 2.4 Distribution of modulated signal with different SNR

2.3 Bit Error Ratio

BER is used to evaluate the effects of noise in this lab including theoretical ber and practical ber. We use Lib SciPy to compute theoretical ber as the following formula shown. And code is shown in Fig. 2.4.

$$\frac{1}{2}\operatorname{erfc}\sqrt{\frac{10^{snr(\text{dB})/10}}{k}}$$

```python
# compute theoretical ber using erfc function
theoryBer = lambda snr, word_len: 0.5 * special.erfc(np.sqrt(10**(snr/10.)/word_len))
```

Figure 2.4 Theoretical Bit Error Ratios using ERFC function

In assignment 1.4, ber is defined as dividing the number of errors by the total number of bits, 8*Npixels. The relation plot of log (BER) vs SNR (2 dB – 12 dB) is shown in the Fig. 2.5 with BPSK and in the Fig. 2.6 with QPSK, but without Parity Check.

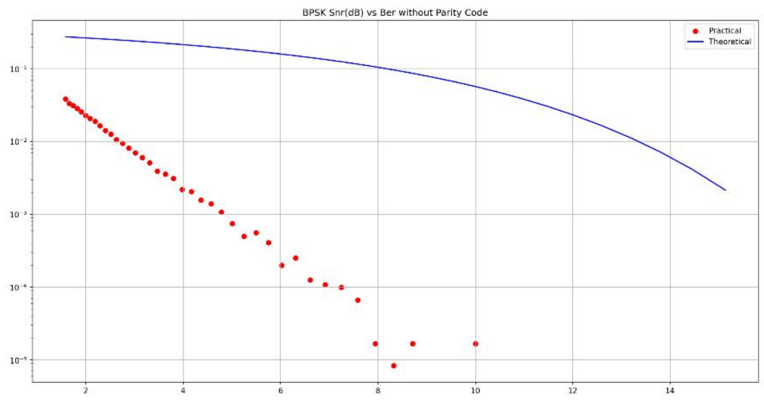


Figure 2.5 BPSK without Parity Check

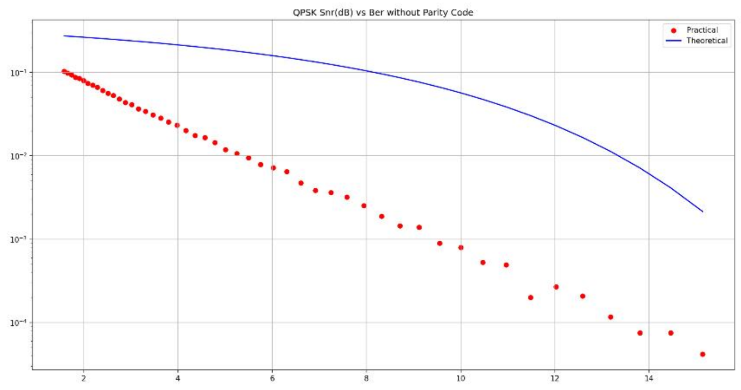


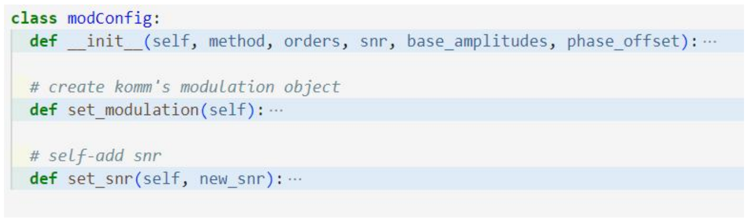
Figure 2.6 QPSK without Parity Check

In assignment 1.5&1.6, ber is defined as the ratio of the total number of Automatic Repeat-reQuest (ARQs) to the number of pixels referring to **3 Parity Check**.

3 Parity Check in PSK & QAM

3.1 Config Modulation

For reuse code, Class modConfig is defined, whose member has method (‘psk’, ‘qam’), orders (2 (2-psk), 4, 16, 256), snr, base_amplitude, phase_offset, modulation object and awgn object, as shown in the Fig. 3.1.

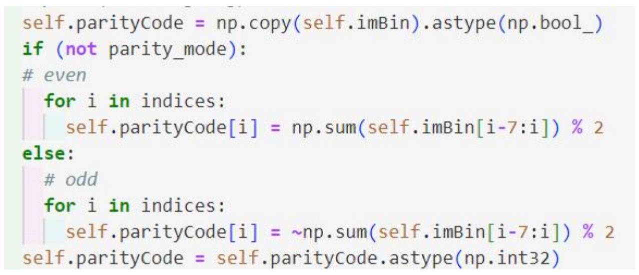


```
class modConfig:
    def __init__(self, method, orders, snr, base_amplitudes, phase_offset):···

    # create komm's modulation object
    def set_modulation(self):···

    # self-add snr
    def set_snr(self, new_snr):···
```

Figure 3.1 Class imgInfo

3.2 Generate Parity Code & Do Parity Check

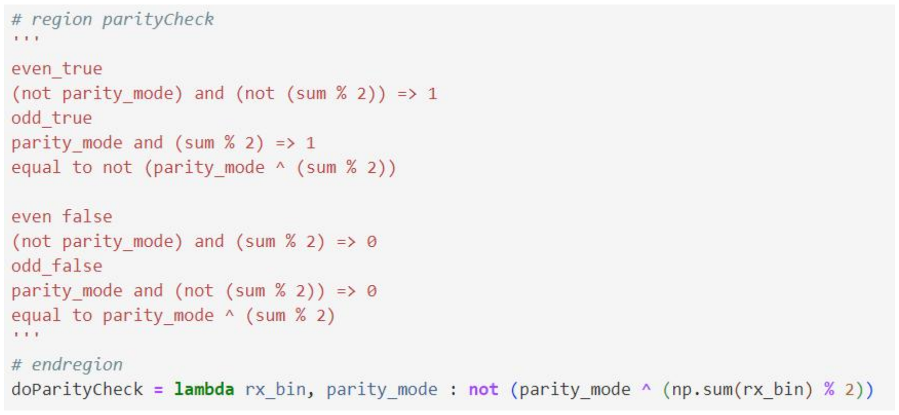
At once time transmitting eight bits, i.e., one pixel, we modifying the **lowest** bit to be the parity code because our eyes cannot distinguish the change of gray value due to the change of the lowest bit. The method createParityCode in Class imgInfo can generate the parity code. The core of code is shown in the Fig. 3.2. It’s noted that we need to cast INT type into BOOL type because of we use NOT operator in the odd field. If not, inversing a ‘0’ of INT type data will not get a ‘1’.



```
self.parityCode = np.copy(self.imBin).astype(np.bool_)
if (not parity_mode):
# even
    for i in indices:
        self.parityCode[i] = np.sum(self.imBin[i-7:i]) % 2
else:
    # odd
    for i in indices:
        self.parityCode[i] = ~np.sum(self.imBin[i-7:i]) % 2
self.parityCode = self.parityCode.astype(np.int32)
```

Figure 3.2 Method createParityCode

Through the analysis of the logical expression, we can use **single statement** to implement the check operation as shown in the Fig. 3.3.



```
# region parityCheck
'''
even_true
(not parity_mode) and (not (sum % 2)) => 1
odd_true
parity_mode and (sum % 2) => 1
equal to not (parity_mode ^ (sum % 2))

even false
(not parity_mode) and (sum % 2) => 0
odd_false
parity_mode and (not (sum % 2)) => 0
equal to parity_mode ^ (sum % 2)
'''
# endregion
doParityCheck = lambda rx_bin, parity_mode : not (parity_mode ^ (np.sum(rx_bin) % 2))
```

Figure 3.3 Method doParityCheck

3.2 PSK & QAM modulation with Parity Check

The final step is that we call the encapsulated function above to stimulate PSK and QAM modulation with differential SNR. We can plot the ratio of the total number of ARQs to the number of pixels from the Fig. 3.4 to 3.11. The Bule line represents theoretical Ber; the red scattered points represent practical Ber. The x-axis represents the changes of SNR from 2 dB to 12 dB.

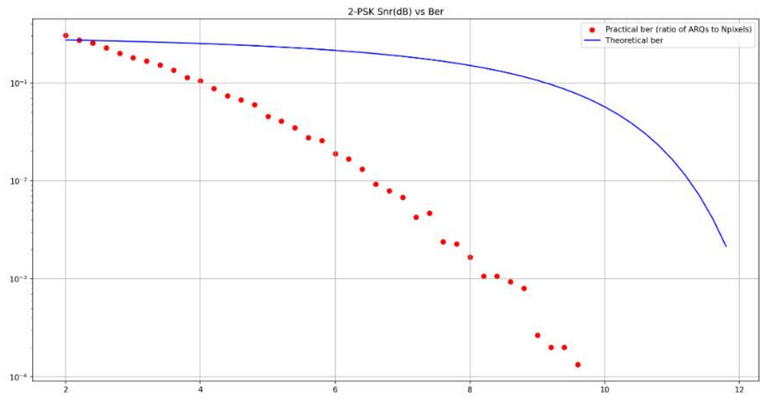


*Figure 3.4 **BPSK** with Parity Check*

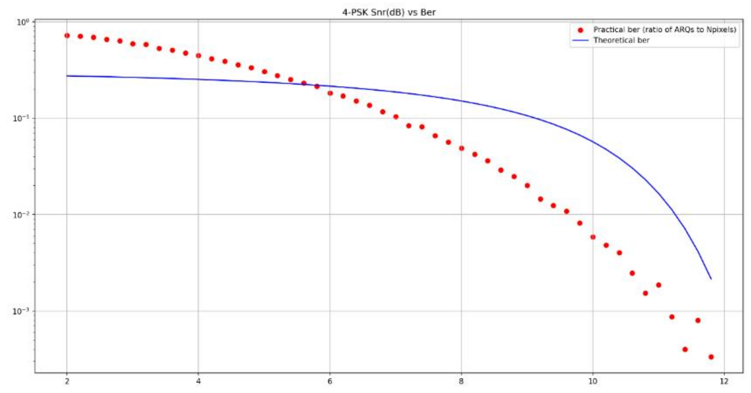


*Figure 3.5 **QPSK** with Parity Check*

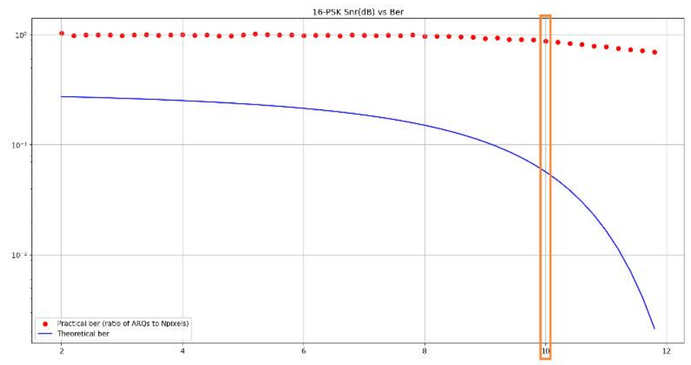


*Figure 3.6 **16-PSK** with Parity Check*

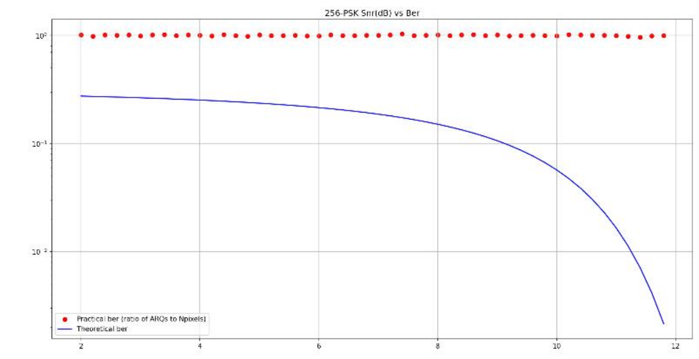


Figure 3.7 ***256-PSK*** *with Parity Check*

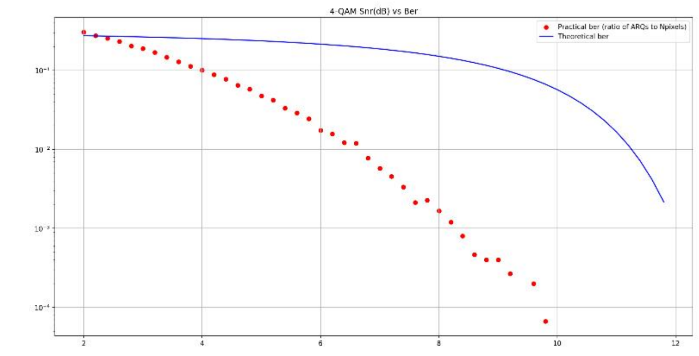


Figure 3.8 ***4-QAM*** *with Parity Check*

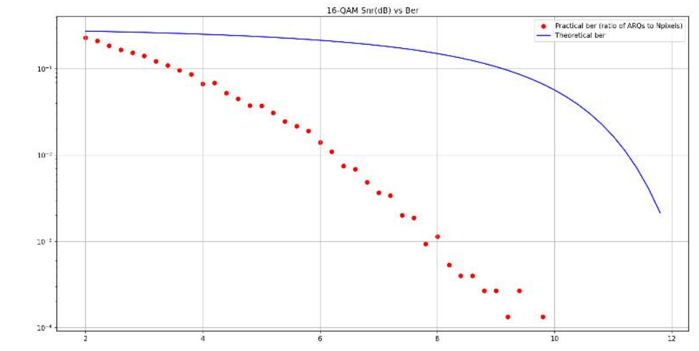


Figure 3.9 ***16-QAM*** *with Parity Check*

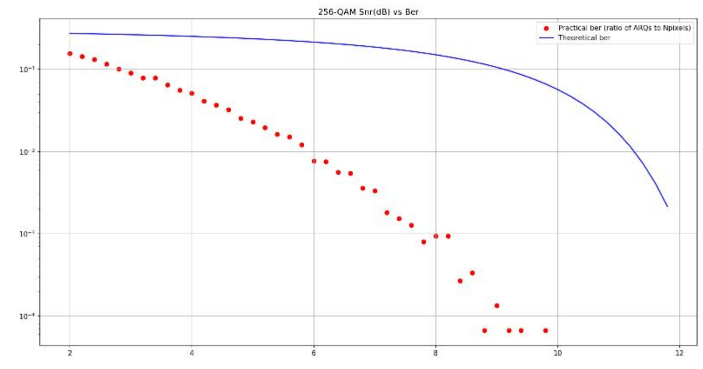


Figure 3.10 ***256-QAM*** *with Parity Check*

4 Conclusion

4.1 PSK vs QAM

PSK modulation only use phase to transmit information. QAM modulation is a compound modulation technique, which combines Amplitude and Phase modulation. The amplitude and phase of modulated signal by QAM contains original information which means carrying more information. Comparing the Fig. 3.5 (**QPSK**) to the Fig. 3.8 (**4-QAM**), with the same noise and transmission bandwidth, QAM has stronger anti-interference capability and carries more information. But QPSK is susceptible to noise. Analyzing the constellation diagram of PSK and QAM can obtain the same conclusion.

4.2 Effects of AWGN

The increment of SNR means the stronger AWGN. With the same modulation method and the number of orders, taking 4-QAM as an example in the Fig. 4.1, bit-error-ratio is negatively correlated with SNR.

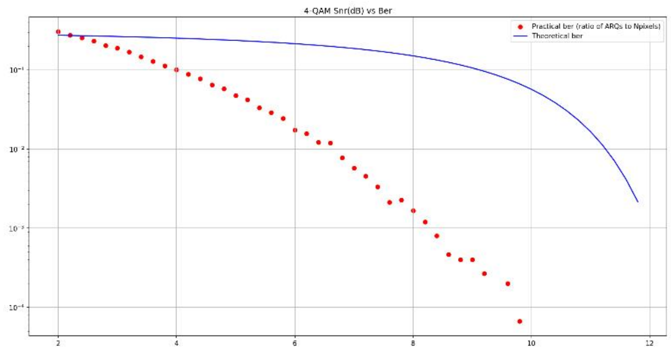


Figure 4.1 ***4-QAM*** *with Parity Check*

Additionally, for PSK modulation, the higher the order, the more likely it is to make errors and receive noise interference. In other words, if we want to get a greater transmission accuracy, the higher the orders, the higher threshold of the signal-to-noise ratio needs to be stronger constrained as shown in the Fig. 3.6, where the orange box field is the threshold of SNR.

4.3 Limitation of Parity Code

Comparing to the Fig. 3.4, 3.5, 3.8, 3.9, 3.10, the relation plot shows the Ber is much lower than theoretical Ber in the most SNR areas. We can draw the conclusion that when using parity check, even passing the checking at the receiving end, the demodulated data may not be the same as the original information. So parity check is simply implemented, but its checking capability is very limited.

Appendix:

AWGNChannel.py

```
'''
Author           : Eureke
Date             : 2023-02-13 10:28:40
LastEditors    : Marcus Wong
LastEditTime : 2023-02-22 23:04:42
Description    : AWGN Channel
'''
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
import komm
from scipy import special
from ParityCheck import imgInfo, theoryBer


# 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


if __name__ == '__main__':
    # 注意当前打开的工作路径的相对路径，这里打开的文件夹是 LAB，所以要加子文件夹
路径 ./Lab3/
    is_plot = True # True
    # open image and binary information
    fp = "./Lab3/DC4_150x100.pgm"
    word_len = 8 # 256 bits per pixel
    img = imgInfo(fp, word_len)

    # save ber ,theoretical ber and snr
    ber_out = np.empty(0)
    theory_ber_out = np.empty(0)
    snr_out = np.empty(0)

    for i in np.arange(2., 12., 0.2):
        # BPSK modulation
        psk = komm.PSKModulation(4)
        # Additive white gaussian noise (AWGN) channel
        # snr -> signal-to-noise ratio
        # snr = np.array([10**(6./10.), 10**(5./10.), 10**(4./10.), 10**(3./10.), 10**(2./10.),
10**(1./10.)])
        awgn = komm.AWGNChannel(snr=10**(i/10.))
        tx_data = psk.modulate(img.imBin)
        # demodulation
        rx_data = awgn(tx_data)
        if (not is_plot):
            # is_plot = not is_plot
            plt.figure()
            plt.axes().set_aspect("equal")
            plt.title("Modulated signal with noise")
            plt.scatter(rx_data[:10000].real,rx_data[:10000].imag,s=1,marker=".")
            plt.show()
        rx_bin = psk.demodulate(rx_data)

        if (False):
            img.displayDemodImage(rx_bin.astype(np.bool_))

        snr = 10**(i/10.)
        ber = practiceBer(img.imBin, rx_bin)
        theory_ber = theoryBer(snr, img.word_len)

        snr_out = np.append(snr_out, snr)
        ber_out = np.append(ber_out, ber)
        theory_ber_out = np.append(theory_ber_out, theory_ber)

        print("snr :", snr)
        print('ber practice: {:.30}%'.format(ber * 100.0 ))
        print('ber real: {:.3}%'.format(theory_ber * 100.0 ))


    if (True):
        plt.figure()
        plt.title("QPSK Snr(dB) vs Ber without Parity Code")
        plt.scatter(snr_out, ber_out, color='r', label='Practical')
        plt.plot(snr_out, theory_ber_out, color='b', label='Theoretical')
        plt.yscale("log")
        plt.grid(True)
        plt.legend()
        plt.show()
```

ParityCheck.py

```
'''
Author          : Eureke
Date            : 2023-02-17 11:49:10
LastEditors   : Marcus Wong
LastEditTime : 2023-02-22 22:04:22
Description   : ParityCheck
'''
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
import komm
from scipy import special

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

    # 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

    # use parity code before transmission
    def createParityCode(self, parity_mode):
        '''
        parity: choose even(0) or odd(1)
        '''
        indices = np.arange(self.word_len - 1, self.imBin.size, self.word_len, dtype=int)
        # print(indices[10:])
        self.parityCode = np.copy(self.imBin).astype(np.bool_)
        if (not parity_mode):
        # even
            for i in indices:
                self.parityCode[i] = np.sum(self.imBin[i-7:i]) % 2
        else:
            # odd
        for i in indices:
            self.parityCode[i] = ~np.sum(self.imBin[i-7:i]) % 2
        self.parityCode = self.parityCode.astype(np.int32)
        return self.parityCode

    # show demodulated image
    def displayDemodImage(self, rx_bin):
        # demod signal with noise
        rx_im = np.packbits(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()


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()


# simulate transmitter send single original word
tx_ori = lambda signal, word_len, start_index : signal[start_index: start_index + word_len]
```

```
# region parityCheck
'''
even_true
(not parity_mode) and (not (sum % 2)) => 1
odd_true
parity_mode and (sum % 2) => 1
equal to not (parity_mode ^ (sum % 2))

even false
(not parity_mode) and (sum % 2) => 0
odd_false
parity_mode and (not (sum % 2)) => 0
equal to parity_mode ^ (sum % 2)
'''
# endregion
doParityCheck = lambda rx_bin, parity_mode : not (parity_mode ^ (np.sum(rx_bin) % 2))


# simulate receiver demodluate signal
def rx_sim(s_mod, mod_config):
    rx_data = mod_config.awgn(s_mod)
    rx_bin = mod_config.modulation.demodulate(rx_data)
    # print(rx_bin)
    return rx_bin


# stimulate transmit single img
def transmission(img, mod_config, parity_mode):
    # save checked demodulation signal
    rx_bin = np.empty(0)
    # arq counter
    arq_cnt = 0

    # simulate single step of transmission
    hasTransPixel = 0 # pixel number has been trasmitted
    while hasTransPixel < img.Npixels:
        # original signal per step
        tx_single = tx_ori(img.parityCode, img.word_len, hasTransPixel * img.word_len)
        # modulation, transmit single word
        s_mod = mod_config.modulation.modulate(tx_single)
        # receive and demodulate single word
        rx_single = rx_sim(s_mod, mod_config)
        # print('tx_single: ', tx_single)
        # print('tx_single size: ', tx_single.size)
        # print('s_mod: ', s_mod)
        # print('s_mod size: ', s_mod.size)
        # print('rx_single: ', rx_single)
        # print('rx_single size: ', rx_single.size)

        # judge transmission error
        if doParityCheck(rx_single, parity_mode): # no error
            hasTransPixel += 1
            rx_bin = np.append(rx_bin, rx_single)
            # print("pass")
        else: # error, repeat transmit
            arq_cnt += 1
            # print('error')
    # bit error ratio
    ber = arq_cnt / img.Npixels
    # print('arq counter: ', arq_cnt)
    # print('Npixels: ', Npixels)
    print('snr(dB): ', mod_config.snr)
    print('bit error ratio: {:.3}%'.format(ber * 100.0 ))
    if (False):
        img.displayDemodImage(rx_bin.astype(np.bool_))

    return ber, mod_config.snr

# compute theoretical ber using erfc function
theoryBer = lambda snr, word_len: 0.5 * special.erfc(np.sqrt(10**(snr/10.)/word_len))

def repeatTransmit(img, parity_mode, method, orders, snr_ctrl, base_amplitudes=1., phase_offset=0.):
    print("Start " + str(orders) + '-' + method + "modulation:")

    # create parity code
    img.createParityCode(parity_mode)
    # print("original bin: ", img.imBin[88:104])
    # print("parity code: ", img.parityCode[88:104])

    # initial modulation config
    # snr = 10**(6./10.) # dB(信噪比强度) = 10*lg(signal/noise) = 6 => signal/noise  约为  3-4 倍
    mod_config = modConfig(method, orders, snr_ctrl[0], base_amplitudes, phase_offset)

    # save ber and snr of each trasmission single image
    ber_out = np.empty(0)
    theory_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)
        ber, snr = transmission(img, mod_config, parity_mode)
        theory_ber = theoryBer(snr, img.word_len)

        ber_out = np.append(ber_out, ber)
        theory_ber_out = np.append(theory_ber_out, theory_ber)
        snr_out = np.append(snr_out, i)
    # print(ber_out)
    # print(theory_ber_out)
    # print(snr_out)

    if (True):
        plt.figure()
        plt.title(str(orders) + '-' + method.upper() + " Snr(dB) vs Ber")
        plt.scatter(snr_out, ber_out, color='r', label='Practical ber (ratio of ARQs to Npixels)')
        plt.plot(snr_out, theory_ber_out, color='b', label='Theoretical ber')
        plt.yscale("log")
        plt.grid(True)
        plt.legend()
        plt.show()


if __name__ == '__main__':
    # open image and binary information
    fp = './Lab3/DC4_150x100.pgm'
    # fp = './Lab3/DC4_640x480.pgm'
    word_len = 8 # 256 bits per pixel
    img = imgInfo(fp, word_len)
    # use parity mode even(0), odd(1)
    parity_mode = 0
    snr_ctrl = [2., 12., 0.2]
    # psk modulation
    repeatTransmit(img=img, parity_mode=parity_mode, method='psk', orders=2, snr_ctrl=snr_ctrl,
base_amplitudes=1., phase_offset=0.)
    repeatTransmit(img=img, parity_mode=parity_mode, method='psk', orders=4, snr_ctrl=snr_ctrl,
base_amplitudes=1., phase_offset=0.)
    repeatTransmit(img=img, parity_mode=parity_mode, method='psk', orders=16,
snr_ctrl=snr_ctrl, base_amplitudes=1., phase_offset=0.)
    repeatTransmit(img=img, parity_mode=parity_mode, method='psk', orders=256,
snr_ctrl=snr_ctrl, base_amplitudes=1., phase_offset=0.)
```

```
  # qam mudulation
  repeatTransmit(img=img, parity_mode=parity_mode, method='qam', orders=4,
snr_ctrl=snr_ctrl, base_amplitudes=1., phase_offset=0.)
  repeatTransmit(img=img, parity_mode=parity_mode, method='qam', orders=16,
snr_ctrl=snr_ctrl, base_amplitudes=1., phase_offset=0.)
  repeatTransmit(img=img, parity_mode=parity_mode, method='qam', orders=256,
snr_ctrl=snr_ctrl, base_amplitudes=1., phase_offset=0.)
```