SUSTech_EE326_Project_Final_Report_Appendix

11911521 钟新宇

1. Introduction

This is an appendix to the final project report for the Spring 2022 EE326 Digital Image Processing course at SUSTech This document contains the code involved in the project.

Outline

2. Base Functions

```python
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :Filter_Base.py
# @Time        :2022-05-23 16:30
# @Author     :钟新宇
import numpy as np
from matplotlib import pyplot as plt
import cv2


def dec2bit(num_dec):
      s = bin(num_dec)[2:]
      # print(len(s))
      while len(s) < 8:
            s = "0" + s   # 满足条件的高位补零
      num_bit = np.zeros(8, dtype=int)
      for o in range(8):
            num_bit[o] = s[o]
      return num_bit


def bit2dec(num_bit):
      arr = np.asarray(num_bit, dtype=str)
      s = "0b" + "".join(arr)
      num_dec = int(s, 2)
      return num_dec

```

```
28
29  def readimg(path):
30      img_in = plt.imread(path)
31      img_in = np.asarray(img_in, dtype=int)
32      showimg(img_in, title="img_in")
33      img_size = img_in.shape
34      img_r = img_in[:, :, 0]
35      img_g = img_in[:, :, 1]
36      img_b = img_in[:, :, 2]
37
38      img_r_show = np.zeros(img_size, dtype=int)
39      img_r_show[:, :, 0] = img_r
40      # showimg(img_r_show, title="img_r")
41      img_g_show = np.zeros(img_size, dtype=int)
42      img_g_show[:, :, 1] = img_g
43      # showimg(img_g_show, title="img_g")
44      img_b_show = np.zeros(img_size, dtype=int)
45      img_b_show[:, :, 2] = img_b
46      # showimg(img_b_show, title="img_b")
47      return img_in, img_r, img_g, img_b
48
49
50  def showimg(img, title="Image"):
51      plt.figure()
52      plt.title(title)
53      plt.imshow(img)
54      plt.show()
55
56
57  def saveimg(img_rgb, path):
58      path_str = np.str(path)
59      bgr = rgb2bgr(img_rgb)
60      cv2.imwrite(filename=path_str, img=bgr)
61
62
63  def img2rgb(rgb):
64      img = np.asarray(rgb, dtype=int)
65      r = img[:, :, 0]
66      g = img[:, :, 1]
67      b = img[:, :, 2]
68      return r, g, b
69
70
71  def rgb2img(size, r, g, b):
72      m, n = size
73      rgb = np.zeros((m, n, 3), dtype=int)
74      rgb[:, :, 0] = r
75      rgb[:, :, 1] = g
76      rgb[:, :, 2] = b
77      return rgb
78
79
80  def rgb2bgr(rgb):
81      rgb = np.asarray(rgb, dtype=int)
82      bgr = np.zeros(rgb.shape, dtype=int)
83      bgr[:, :, 0] = rgb[:, :, 2]
84      bgr[:, :, 1] = rgb[:, :, 1]
85      bgr[:, :, 2] = rgb[:, :, 0]
```

```
      return bgr


if __name__ == '__main__':
      try:
            pass
      except KeyboardInterrupt:
            pass
```

3. Baker Transform

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :baker.py
# @Time        :2022-05-23 18:31
# @Author     :钟新宇
import numpy as np


def baker_stretch(array):
      """
     Program a baker_stretch(array) function that returns the new array obtained by
“stretching”
     the input table.

     Stretching. The principle is as follows: the first two lines (each with a length of n)
produce a single
     line with a length of 2n. We mix the values of each line by alternating an upper element
and a
     lower element.

     Formulas. An element at position (i, j) of the target array, corresponds to an element
(2i, j//2) (if j
     is even) or (2i + 1, j//2) (if j is odd) of the source array, with here 0 ⩽ i < n
     2 and 0 ⩽ j < 2n.

     :param array: The input array
     :return: The output array after baker_stretch
     """
      array = np.asarray(array, dtype=int)
      m, n = array.shape
      array_out = np.zeros((int(m / 2), int(n * 2)), dtype=int)
      for i in range(int(m / 2)):
            for j in range(int(n * 2)):
                  if j % 2 == 0:
                        array_out[i, j] = array[2 * i, j // 2]
                  else:
                        array_out[i, j] = array[2 * i + 1, j // 2]
      # print("strtch")
      # print(array_out)
      return array_out


def baker_fold(array):
```

```
    """
    Program a baker_fold(array) function that returns the table obtained by “folding” the input
    table.

    Fold. The principle is as follows: the right part of a stretched array is turned upside down, then
    added under the left part. Starting from a n/2 × 2n array you get an n × n array.

    Formulas. For 0 ⩽ i < n/2 and 0 ⩽ j < n the elements in position (i, j) of the array are kept in
    place. For n/2 ⩽ i < n and 0 ⩽ j < n an element of the array (i, j) corresponds to an element
    (n/2 − i − 1,2n − 1 − j) of the source array.

    :param array: The input array after baker_stretch
    :return: The output array after baker_fold
    """
    array = np.asarray(array, dtype=int)
    m, n = array.shape
    array_out = np.zeros((int(m * 2), int(n / 2)), dtype=int)
    for i in range(0, m):
        for j in range(int(n / 2)):
            array_out[i, j] = array[i, j]
    for i in range(m, m * 2):
        for j in range(int(n / 2)):
            array_out[i, j] = array[m - i - 1, n - 1 - j]
    # print("fold")
    # print(array_out)
    return array_out


def baker_iterate(array, k):
    """
    Program a baker_iterate(array,k) function that returns the table calculated after k iterations
    of baker’s transformation.

    Caution! It sometimes takes many iterations to get back to the original image. For example when
    n = 4, we return to the starting image after k = 5 iterations; when n = 256 it takes k = 17.
    Conjecture a return value in the case where n is a power of 2. However, for n = 10, you need
    k = 56920 iterations!

    :param array:
    :param k:
    :return:
    """
    for i in range(k):
        print("iterate No.%d" % (i + 1))
        array = baker(array)
        print(array)

    return array
```

```
def baker(array):
	array_stretch = baker_stretch(array)
	array_fold = baker_fold(array_stretch)
	return array_fold


if __name__ == '__main__':
	try:
		array_in = np.asarray([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]], dtype=int)
		print(array_in)
		# array_baker = baker(array_in)
		array_baker = baker_iterate(array_in, k=5)
	except KeyboardInterrupt:
		pass
```

4. LSB

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :wm_lsb.py
# @Time      :2022-05-23 12:49
# @Author    :钟新宇

import numpy as np
from Base import readimg, showimg, saveimg, rgb2img, rgb2bgr, dec2bit, bit2dec
from baker import baker_iterate


# def enc(mark, k):
#     return baker_iterate(mark, k=k)


def enc(mark_r, mark_g, mark_b, k):
	mark_r = baker_iterate(mark_r, k=k)
	mark_g = baker_iterate(mark_g, k=k)
	mark_b = baker_iterate(mark_b, k=k)
	return mark_r, mark_g, mark_b


def encode(path, key):
	img_src_path, img_mark_path, mark_src_path, mark_enc_path = path

	# reading data
	img_src, img_r, img_g, img_b = readimg(path=img_src_path)
	mark_src, mark_r, mark_g, mark_b = readimg(path=mark_src_path)

	# get size
	img_src_m, img_src_n, _ = img_src.shape
	mark_m, mark_n, _ = mark_src.shape
	if mark_m * mark_n * 8 > img_src_m * img_src_n:
		print("The source image is too small")

	# dec2bit
```

```
        bin_mark_r = np.zeros((mark_m, mark_n, 8), dtype=int)
        bin_mark_g = np.zeros((mark_m, mark_n, 8), dtype=int)
        bin_mark_b = np.zeros((mark_m, mark_n, 8), dtype=int)
        for i in range(mark_m):
                for j in range(mark_n):
                        bin_mark_r[i, j, :] = dec2bit(mark_r[i, j])
                        bin_mark_g[i, j, :] = dec2bit(mark_g[i, j])
                        bin_mark_b[i, j, :] = dec2bit(mark_b[i, j])
        bin_mark_num = mark_m * mark_n * 8
        bin_mark_r2 = np.reshape(bin_mark_r, bin_mark_num)
        bin_mark_g2 = np.reshape(bin_mark_g, bin_mark_num)
        bin_mark_b2 = np.reshape(bin_mark_b, bin_mark_num)

        # padding
        pad_num = img_src_m * img_src_n - mark_m * mark_n * 8
        bin_mark_r2 = np.pad(bin_mark_r2, (0, pad_num), mode='wrap')
        bin_mark_g2 = np.pad(bin_mark_g2, (0, pad_num), mode='wrap')
        bin_mark_b2 = np.pad(bin_mark_b2, (0, pad_num), mode='wrap')

        bin_mark_r2 = np.reshape(bin_mark_r2, (img_src_m, img_src_n))
        bin_mark_g2 = np.reshape(bin_mark_g2, (img_src_m, img_src_n))
        bin_mark_b2 = np.reshape(bin_mark_b2, (img_src_m, img_src_n))

        # mark enc
        bin_mark_r2, bin_mark_g2, bin_mark_b2 = enc(bin_mark_r2, bin_mark_g2, bin_mark_b2,
k=key)

        bin_mark_out_rgb = rgb2img(size=(img_src_m, img_src_n), r=bin_mark_r2, g=bin_mark_g2,
b=bin_mark_b2)
        bin_mark_out_rgb[bin_mark_out_rgb[:, :, 0] == 1] = 255
        bin_mark_out_rgb[bin_mark_out_rgb[:, :, 1] == 1] = 255
        bin_mark_out_rgb[bin_mark_out_rgb[:, :, 2] == 1] = 255
        showimg(bin_mark_out_rgb, title="bin_mark_rgb")
        saveimg(bin_mark_out_rgb, path=mark_enc_path)

        # lsb
        bin_src_r = np.zeros((img_src_m, img_src_n, 8), dtype=int)
        bin_src_g = np.zeros((img_src_m, img_src_n, 8), dtype=int)
        bin_src_b = np.zeros((img_src_m, img_src_n, 8), dtype=int)
        dec_src_r = np.zeros((img_src_m, img_src_n), dtype=int)
        dec_src_g = np.zeros((img_src_m, img_src_n), dtype=int)
        dec_src_b = np.zeros((img_src_m, img_src_n), dtype=int)

        for i in range(img_src_m):
                for j in range(img_src_n):
                        bin_src_r[i, j] = dec2bit(img_r[i, j])
                        bin_src_g[i, j] = dec2bit(img_g[i, j])
                        bin_src_b[i, j] = dec2bit(img_b[i, j])
                        bin_src_r[i, j, 7] = bin_mark_r2[i, j]
                        bin_src_g[i, j, 7] = bin_mark_g2[i, j]
                        bin_src_b[i, j, 7] = bin_mark_b2[i, j]
                        dec_src_r[i, j] = bit2dec(bin_src_r[i, j, :])
                        dec_src_g[i, j] = bit2dec(bin_src_g[i, j, :])
                        dec_src_b[i, j] = bit2dec(bin_src_b[i, j, :])

        # output
        img_out_rgb = rgb2img(size=(img_src_m, img_src_n), r=dec_src_r, g=dec_src_g,
b=dec_src_b)
```

```
92        showimg(img_out_rgb, title="image with mark")
93        saveimg(img_out_rgb, path=img_mark_path)
94        return np.asarray((mark_m, mark_n), dtype=int)
95
96
97  def decode(path, size_mark: np.ndarray, key: int):
98        img_mark_path, mark_rec_path = path
99
100       # read file
101       img_rec, img_rec_r, rec_g, rec_b = readimg(path=img_mark_path)
102       # get size
103       rec_m, rec_n, _ = img_rec.shape
104       mark_m, mark_n = size_mark
105
106       # dec2bit
107       bin_rec_r = np.zeros((rec_m, rec_n, 8), dtype=int)
108       bin_rec_g = np.zeros((rec_m, rec_n, 8), dtype=int)
109       bin_rec_b = np.zeros((rec_m, rec_n, 8), dtype=int)
110       bin_mark_rec_r = np.zeros((rec_m, rec_n), dtype=int)
111       bin_mark_rec_g = np.zeros((rec_m, rec_n), dtype=int)
112       bin_mark_rec_b = np.zeros((rec_m, rec_n), dtype=int)
113
114       for i in range(rec_m):
115             for j in range(rec_n):
116                   bin_rec_r[i, j, :] = dec2bit(img_rec_r[i, j])
117                   bin_rec_g[i, j, :] = dec2bit(rec_g[i, j])
118                   bin_rec_b[i, j, :] = dec2bit(rec_b[i, j])
119                   bin_mark_rec_r[i, j] = bin_rec_r[i, j, 7]
120                   bin_mark_rec_g[i, j] = bin_rec_g[i, j, 7]
121                   bin_mark_rec_b[i, j] = bin_rec_b[i, j, 7]
122
123       bin_mark_rec_r, bin_mark_rec_g, bin_mark_rec_b = enc(bin_mark_rec_r, bin_mark_rec_g,
    bin_mark_rec_b, k=key)
124
125       img_rec_size = rec_m * rec_n
126       bin_mark_size = mark_m * mark_n * 8
127       mark_pad_num = img_rec_size / bin_mark_size
128
129       bin_mark_rec_r = np.reshape(bin_mark_rec_r, img_rec_size)
130       bin_mark_rec_g = np.reshape(bin_mark_rec_g, img_rec_size)
131       bin_mark_rec_b = np.reshape(bin_mark_rec_b, img_rec_size)
132
133       # bin_mark_rec_r1 = np.zeros((mark_pad_num, mark_m, mark_n, 8), dtype=int)
134       # bin_mark_rec_g1 = np.zeros((mark_pad_num, mark_m, mark_n, 8), dtype=int)
135       # bin_mark_rec_b1 = np.zeros((mark_pad_num, mark_m, mark_n, 8), dtype=int)
136       # for i in mark_pad_num:
137       #       bin_mark_rec_r1[i, :, :] = np.reshape(bin_mark_rec_r[i * bin_mark_size:(i + 1) *
    bin_mark_size],
138       #                                             (mark_m, mark_n, 8))
139       #       bin_mark_rec_g1[i, :, :] = np.reshape(bin_mark_rec_g[i * bin_mark_size:(i + 1) *
    bin_mark_size],
140       #                                             (mark_m, mark_n, 8))
141       #       bin_mark_rec_b1[i, :, :] = np.reshape(bin_mark_rec_b[i * bin_mark_size:(i + 1) *
    bin_mark_size],
142       #                                             (mark_m, mark_n, 8))
143
144       bin_mark_rec_r1 = bin_mark_rec_r[0:bin_mark_size]
145       bin_mark_rec_g1 = bin_mark_rec_g[0:bin_mark_size]
```

```
    bin_mark_rec_b1 = bin_mark_rec_b[0:bin_mark_size]

    bin_mark_rec_r1 = np.reshape(bin_mark_rec_r1, (mark_m, mark_n, 8))
    bin_mark_rec_g1 = np.reshape(bin_mark_rec_g1, (mark_m, mark_n, 8))
    bin_mark_rec_b1 = np.reshape(bin_mark_rec_b1, (mark_m, mark_n, 8))

    mark_rec_m, mark_rec_n = mark_m, mark_n
    dec_mark_rec_r = np.zeros((mark_rec_m, mark_rec_n), dtype=int)
    dec_mark_rec_g = np.zeros((mark_rec_m, mark_rec_n), dtype=int)
    dec_mark_rec_b = np.zeros((mark_rec_m, mark_rec_n), dtype=int)
    for i in range(mark_rec_m):
        for j in range(mark_rec_n):
            dec_mark_rec_r[i, j] = bit2dec(bin_mark_rec_r1[i, j, :])
            dec_mark_rec_g[i, j] = bit2dec(bin_mark_rec_g1[i, j, :])
            dec_mark_rec_b[i, j] = bit2dec(bin_mark_rec_b1[i, j, :])

    mark_rec_rgb = rgb2img(size=(mark_rec_m, mark_rec_n), r=dec_mark_rec_r,
g=dec_mark_rec_g, b=dec_mark_rec_b)
    showimg(mark_rec_rgb)
    saveimg(mark_rec_rgb, path=mark_rec_path)


if __name__ == "__main__":
    # key: 1024: 2^10, baker 21 times
    k1 = 10
    k2 = 11
    """
   src path
        "./img/img_src.bmp",  # img_src_path
        "./img/img_mark.bmp"  # img_mark_path
        "./img/mark_src.bmp",  # mark_src_path
        "./img/mark_enc.bmp",  # mark_enc_path
        "./img/mark_rec.bmp"  # mark_rec_path
   """

    # # mark size
    # mark_size = np.asarray([256, 256], dtype=int)

    # mark_size = encode(path=["./img/img_src.bmp", "./img/img_mark.bmp",
"./img/mark_src.bmp", "./img/mark_enc.bmp"],
    #                    key=k1)
    # decode(path=["./img/img_mark.bmp", "./img/mark_rec.bmp"], size_mark=mark_size, key=k2)

    img_src, _, _, _ = readimg("./img/img_src.bmp")
    img_enc, _, _, _ = readimg("./img/img_mark.bmp")
    img_diff = img_src - img_enc
    # img_diff[img_diff[:, :, :] == 1] = 255
    showimg(img_diff)
    saveimg(img_diff, path="./img/img_diff.bmp")

```

5. Attack Test

Note: I used the filters designed in the last assignment and called them directly during the attack test.

```
from lib.MeanFilters import arithmetic_mean
from lib.OrderStatisticFilters import median_filter
```

This document does not contain code about them.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :attack.py
# @Time      :2022-05-24 21:01
# @Author    :钟新宇

import numpy as np
from matplotlib import pyplot as plt
from Base import showimg, saveimg, img2rgb, rgb2img
from lib.MeanFilters import arithmetic_mean
from lib.OrderStatisticFilters import median_filter
from wm_lsb import decode
import cv2
import random


def psnr(src, rec):
    # read image
    img_src, img_rec = np.asarray(src, dtype=int), np.asarray(rec, dtype=int)
    # get size
    m, n, _ = img_src.shape
    # get rgb component
    src_r, src_g, src_b = img2rgb(img_src)
    rec_r, rec_g, rec_b = img2rgb(img_rec)
    # calculate error
    error_r, error_g, error_b = np.abs(np.subtract(src_r, rec_r)), np.abs(np.subtract(src_g, rec_g)), np.abs(np.subtract(src_b, rec_b))
    # calculate sum
    sum_r, sum_g, sum_b = np.sum(error_r), np.sum(error_g), np.sum(error_b)
    # calculate mse
    mse_r, mse_g, mse_b = np.multiply(sum_r, 1 / (m * n)), np.multiply(sum_g, 1 / (m * n)), np.multiply(sum_b, 1 / (m * n))
    # find maximum
    max_r, max_g, max_b = np.max(src_r), np.max(src_g), np.max(src_b)
    # calculate snr
    psnr_r = 10 * np.log10(max_r ** 2 / mse_r) if mse_r != 0 else -np.inf
    psnr_g = 10 * np.log10(max_g ** 2 / mse_g) if mse_g != 0 else -np.inf
    psnr_b = 10 * np.log10(max_b ** 2 / mse_b) if mse_b != 0 else -np.inf
    snr = np.asarray([psnr_r, psnr_g, psnr_b], dtype=float)
    print(snr)
    return snr


def nc(src, rec):
    # read image
    mark_src = np.asarray(src, dtype=int)
    mark_rec = np.asarray(rec, dtype=int)
    # get size
    m, n, _ = mark_src.shape
```

```
    # get rgb component
    src_r, src_g, src_b = img2rgb(mark_src)
    rec_r, rec_g, rec_b = img2rgb(mark_rec)
    # calculate sum1
    sum1_r = np.sum(np.multiply(src_r, rec_r))
    sum1_g = np.sum(np.multiply(src_g, rec_r))
    sum1_b = np.sum(np.multiply(src_b, rec_r))
    # calculate sum2 (src)
    sum2_r = np.sqrt(np.sum(np.power(src_r, 2)))
    sum2_g = np.sqrt(np.sum(np.power(src_g, 2)))
    sum2_b = np.sqrt(np.sum(np.power(src_b, 2)))
    # calculate sum3 (rec)
    sum3_r = np.sqrt(np.sum(np.power(rec_r, 2)))
    sum3_g = np.sqrt(np.sum(np.power(rec_g, 2)))
    sum3_b = np.sqrt(np.sum(np.power(rec_b, 2)))
    # calculate nc
    nc_r = sum1_r / (sum2_r * sum3_r)
    nc_g = sum1_g / (sum2_g * sum3_g)
    nc_b = sum1_b / (sum2_b * sum3_b)
    nc_out = np.asarray([nc_r, nc_g, nc_b], dtype=float)
    print(nc_out)
    return nc_out


def noise_sp(img, prop):
    img_noise = np.asarray(img, dtype=int)
    m, n, _ = img_noise.shape
    num = int(m * n * prop)
    for i in range(num):
        w = random.randint(0, n - 1)
        h = random.randint(0, m - 1)
        if random.randint(0, 1) == 0:
            img_noise[h, w] = 0
        else:
            img_noise[h, w] = 255
    showimg(img_noise, title="img_noise_sp")
    saveimg(img_noise, path="./img/img_noise_sp.bmp")
    return img_noise


def noise_gaussian(img, mean, sigma):
    img_noise = np.asarray(img, dtype=int)
    noise = np.random.normal(mean, sigma, img_noise.shape)
    img_noise = img_noise + noise
    img_noise = np.clip(img_noise, 0, 255)
    showimg(img_noise, title="img_noise_gaussian")
    saveimg(img_noise, path="./img/img_noise_gaussian.bmp")
    return img_noise


def noise_random(img, noise_num):
    img_noise = np.asarray(img, dtype=int)
    rows, cols, _ = img_noise.shape
    for i in range(noise_num):
        x = np.random.randint(0, rows)   # 随机生成指定范围的整数
        y = np.random.randint(0, cols)
        img_noise[x, y, :] = 255
    showimg(img_noise, title="img_noise_random")
```

```
106       saveimg(img_noise, path="./img/img_noise_random.bmp")
107       return img_noise
108
109
110 def hist_equ(img):
111       img = np.asarray(img, dtype=int)
112       m, n = img.shape
113       L = 256
114       bins = range(L + 1)
115       hist_in, _ = np.histogram(img.flat, bins=bins, density=True)
116
117       s = np.asarray(np.zeros(256))
118       for i in range(L):
119             s[i] = (L - 1) * sum(hist_in[:i + 1])
120
121       img_out = np.asarray(np.zeros((m, n)), dtype=int)
122       for i in range(m):
123             for j in range(n):
124                   img_out[i][j] = s[img[i][j]]
125
126       return img_out
127
128
129 def histogram_equal(img):
130       img_equal = np.asarray(img, dtype=int)
131       m, n, _ = img_equal.shape
132       r, g, b = img2rgb(img_equal)
133       r = np.clip(hist_equ(r), 0, 255)
134       g = np.clip(hist_equ(g), 0, 255)
135       b = np.clip(hist_equ(b), 0, 255)
136       img_equal = rgb2img((m, n), r, g, b)
137       showimg(img_equal, title="img_histogram_equal")
138       saveimg(img_equal, path="./img/img_histogram_equal.bmp")
139       return img_equal
140
141
142 def mean(img):
143       img_filter = np.asarray(img, dtype=int)
144       m, n, _ = img_filter.shape
145       size = 3
146       r, g, b = img2rgb(img_filter)
147       r = arithmetic_mean(r, size=size)
148       g = arithmetic_mean(g, size=size)
149       b = arithmetic_mean(b, size=size)
150       img_filter = rgb2img((m, n), r, g, b)
151       showimg(img_filter, title="img_filter_mean")
152       saveimg(img_filter, path="./img/img_filter_mean.bmp")
153       return img_filter
154
155
156 def median(img):
157       img_filter = np.asarray(img, dtype=int)
158       m, n, _ = img_filter.shape
159       size = 3
160       r, g, b = img2rgb(img_filter)
161       r = median_filter(r, size=size)
162       g = median_filter(g, size=size)
163       b = median_filter(b, size=size)
```

```
    img_filter = rgb2img((m, n), r, g, b)
    showimg(img_filter, title="img_filter_median")
    saveimg(img_filter, path="./img/img_filter_median.bmp")
    return img_filter


if __name__ == '__main__':
    try:
        """
        src
        "./img/img_src.bmp",  # img_src_path
        "./img/img_mark.bmp"  # img_mark_path
        "./img/mark_src.bmp",  # mark_src_path
        "./img/mark_enc.bmp",  # mark_enc_path
        "./img/mark_rec.bmp"  # mark_rec_path

        noise
        "./img/img_noise_sp.bmp"
        "./img/mark_noise_sp.bmp"
        "./img/img_noise_gaussian.bmp"
        "./img/mark_noise_gaussian.bmp"
        "./img/img_noise_random.bmp"
        "./img/mark_noise_random.bmp"

        histogram equalization
        "./img/img_histogram_equal.bmp"
        "./img/mark_histogram_equal.bmp"

        filter
        "./img/img_filter_mean.bmp"
        "./img/mark_filter_mean.bmp"
        "./img/img_filter_median.bmp"
        "./img/mark_filter_median.bmp"
        """

        mark_size = np.asarray([256, 256], dtype=int)

        img_src = np.asarray(plt.imread("./img/img_src.bmp"), dtype=int)
        img_rec = np.asarray(plt.imread("./img/img_mark.bmp"), dtype=int)
        mark_src = np.asarray(plt.imread("./img/mark_src.bmp"), dtype=int)
        mark_rec = np.asarray(plt.imread("./img/mark_rec.bmp"), dtype=int)

        """noise attack"""
        # img_noise_sp = noise_sp(img_rec, prop=0.1)
        # img_noise_gaussian = noise_gaussian(img_rec, mean=10, sigma=255)
        # img_noise_random = noise_random(img_rec, noise_num=100)
        #
        # decode(path=["./img/img_noise_sp.bmp", "./img/mark_noise_sp.bmp"],
size_mark=mark_size, key=11)
        # decode(path=["./img/img_noise_gaussian.bmp", "./img/mark_noise_gaussian.bmp"],
size_mark=mark_size, key=11)
        # decode(path=["./img/img_noise_random.bmp", "./img/mark_noise_random.bmp"],
size_mark=mark_size, key=11)

        """histogram equalization"""
        # img_equal = histogram_equal(img_rec)
        # decode(path=["./img/img_histogram_equal.bmp", "./img/mark_histogram_equal.bmp"],
size_mark=mark_size, key=11)
```

```
            """filter"""
            # img_filter_mean = mean(img_rec)
            # decode(path=["./img/img_filter_mean.bmp", "./img/mark_filter_mean.bmp"], size_mark=mark_size, key=11)
            # img_filter_median = median(img_rec)
            # decode(path=["./img/img_filter_median.bmp", "./img/mark_filter_median.bmp"], size_mark=mark_size, key=11)

            """psnr and nc"""
            psnr_src = psnr(src=img_src, rec=img_src)
            psnr_1 = psnr(src=img_src, rec=img_rec)
            nc_1 = nc(mark_src, mark_rec)
      except KeyboardInterrupt:
            pass

```

6. FFT

Note: I did not present the frequency domain digital watermarking technique in my report. This is because I think my code is flawed. The principle of frequency domain watermarking technique is to use watermark information to replace the spectral coefficients of the carrier. However, in practice, I multiplied the watermark array by a factor K and then added it directly to the high frequency region. If the coefficient K is too large, the carrier will become dark, and if the coefficient K is too small, the carrier will not change significantly, but the watermark in the frequency domain is very inconspicuous. Perhaps I should use smaller coefficients and then perform histogram equalization. But I don't have time to practice this idea.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :wm_fft.py
# @Time      :2022-05-28 9:30
# @Author    :钟新宇
import cv2
import numpy as np
from numpy.fft import fft2, fftshift, ifft2, ifftshift
from Base import readimg, showimg, saveimg, rgb2img, rgb2bgr, dec2bit, bit2dec
from lib.Filter_Base import normalize
from baker import baker_iterate


def add_mark(img, mark, size):
      img = np.asarray(img, dtype=complex)
      img2 = img.copy()
      m, n = np.asarray(size, dtype=int)

      mark = mark * 20
      mark_flip_1 = np.flip(mark, 0)
      mark_flip_2 = np.flip(mark, 1)
      mark_flip_3 = np.flip(mark_flip_1, 1)
      mark_flip_4 = np.flip(mark_flip_1, 0)
      img2[0:n, 0:m] = np.add(img2[0:n, 0:m], mark, dtype=complex)
      img2[-n:, -m:] = np.add(img2[-n:, -m:], mark_flip_3, dtype=complex)
      img2[-n:, 0:m] = np.add(img2[-n:, 0:m], mark_flip_1, dtype=complex)
      img2[0:n:, -m:] = np.add(img2[0:n:, -m:], mark_flip_2, dtype=complex)
      return img2
```

```
29
30
31  def encoder(path):
32      img_src_path, img_mark_path, mark_src_path, mark_enc_path = path
33
34      # reading data
35      img_src, img_r, img_g, img_b = readimg(path=img_src_path)
36      mark_src, mark_r, mark_g, mark_b = readimg(path=mark_src_path)
37
38      # get size
39      img_src_m, img_src_n, _ = img_src.shape
40      mark_m, mark_n, _ = mark_src.shape
41
42      # fft2
43      img_r_fft = fftshift(fft2(img_r))
44      img_g_fft = fftshift(fft2(img_g))
45      img_b_fft = fftshift(fft2(img_b))
46
47      # log magnitude
48      img_r_mag = normalize(np.log(np.abs(img_r_fft)))
49      img_g_mag = normalize(np.log(np.abs(img_g_fft)))
50      img_b_mag = normalize(np.log(np.abs(img_b_fft)))
51      img_src_mag = normalize(rgb2img(size=(img_src_m, img_src_n), r=img_r_mag, g=img_g_mag,
    b=img_b_mag), dtype=int)
52      showimg(img=img_src_mag, title="img_src_mag")
53      saveimg(img_src_mag, path="./img/img_src_fft_mag.bmp")
54
55      # angle
56      img_r_ang = np.angle(img_r_fft)
57      img_g_ang = np.angle(img_g_fft)
58      img_b_ang = np.angle(img_b_fft)
59
60      # add watermark
61      enc_r_fft = add_mark(img_r_fft, mark=mark_r, size=(mark_m, mark_n))
62      enc_g_fft = add_mark(img_g_fft, mark=mark_g, size=(mark_m, mark_n))
63      enc_b_fft = add_mark(img_b_fft, mark=mark_b, size=(mark_m, mark_n))
64      # magnitude
65      enc_r_mag = normalize(np.log(np.abs(enc_r_fft)))
66      enc_g_mag = normalize(np.log(np.abs(enc_g_fft)))
67      enc_b_mag = normalize(np.log(np.abs(enc_b_fft)))
68      enc_mag = normalize(rgb2img(size=(img_src_m, img_src_n), r=enc_r_mag, g=enc_g_mag,
    b=enc_b_mag), dtype=int)
69      showimg(img=enc_mag, title="enc_mag")
70      saveimg(enc_mag, path="./img/enc_fft_mag.bmp")
71      # ifft2
72      enc_r_ifft = (ifft2(enc_r_fft))
73      enc_g_ifft = (ifft2(enc_g_fft))
74      enc_b_ifft = (ifft2(enc_b_fft))
75
76      img_enc_r = np.asarray(np.abs(enc_r_ifft), dtype=int)
77      img_enc_g = np.asarray(np.abs(enc_g_ifft), dtype=int)
78      img_enc_b = np.asarray(np.abs(enc_b_ifft), dtype=int)
79      img_enc = normalize(rgb2img(size=(img_src_m, img_src_n), r=img_enc_r, g=img_enc_g,
    b=img_enc_b), dtype=int)
80      showimg(img=img_enc, title="img_enc")
81      saveimg(img_enc, path=img_mark_path)
82
83
```

```
84  def decoder(path):
85        img_mark_path = path
86
87        # reading data
88        img_enc, enc_r, enc_g, enc_b = readimg(path=img_mark_path)
89
90        # get size
91        img_m, img_n, _ = img_enc.shape
92
93        # fft2
94        enc_r_fft = fftshift(fft2(enc_r))
95        enc_g_fft = fftshift(fft2(enc_g))
96        enc_b_fft = fftshift(fft2(enc_b))
97
98        # log magnitude
99        enc_r_mag = normalize(np.log(np.abs(enc_r_fft)))
100       enc_g_mag = normalize(np.log(np.abs(enc_g_fft)))
101       enc_b_mag = normalize(np.log(np.abs(enc_b_fft)))
102       enc_mag = normalize(rgb2img(size=(img_m, img_n), r=enc_r_mag, g=enc_g_mag, b=enc_b_mag),
dtype=int)
103       showimg(img=enc_mag, title="enc_mag")
104       saveimg(enc_mag, path="./img/img_rec_mag.bmp")
105
106
107 if __name__ == '__main__':
108       try:
109             """
110           src path
111                 "./img/img_src.bmp",  # img_src_path
112                 "./img/img_mark.bmp"  # img_mark_path
113                 "./img/mark_src.bmp",  # mark_src_path
114                 "./img/mark_enc.bmp",  # mark_enc_path
115                 "./img/mark_rec.bmp"  # mark_rec_path
116           """
117            encode_path = np.asarray(["./img/img_src.bmp", "./img/img_mark.bmp",
"./img/mark_src.bmp", "./img/mark_enc.bmp"], dtype=str)
118            decode_path = "./img/img_mark.bmp"
119            encoder(encode_path)
120            decoder(decode_path)
121            print("end")
122       except KeyboardInterrupt:
123             pass
124
```