数字图像处理 FINAL PROJECT

吕鑫 201411212012

选题

8）实现图像的维纳滤波处理：实现对图像进行维纳滤波处理复原模糊图像，可以考虑只采用长宽均为2的幂的灰度图像测试）

算法介绍

当图像仅仅被一个已知的lowpass filter模糊化的时候，逆滤波可以处理图像进行图像恢复，但是逆滤波对于噪声过于敏感，这时就需要维纳滤波来进行消除有噪声的模糊图像，通过找到未污染图像的一个估计，使得它们之间的均方差最小，可以去除噪声，同时清晰化模糊图像。

$$G(u,v)=\frac{H^*(u,v)P_f(u,v)}{|H(u,v)|^2P_f(u,v)+P_n(u,v)}$$

其中H*是系统传递函数的复共轭

P*f和P*n分别为信号和噪声的功率谱

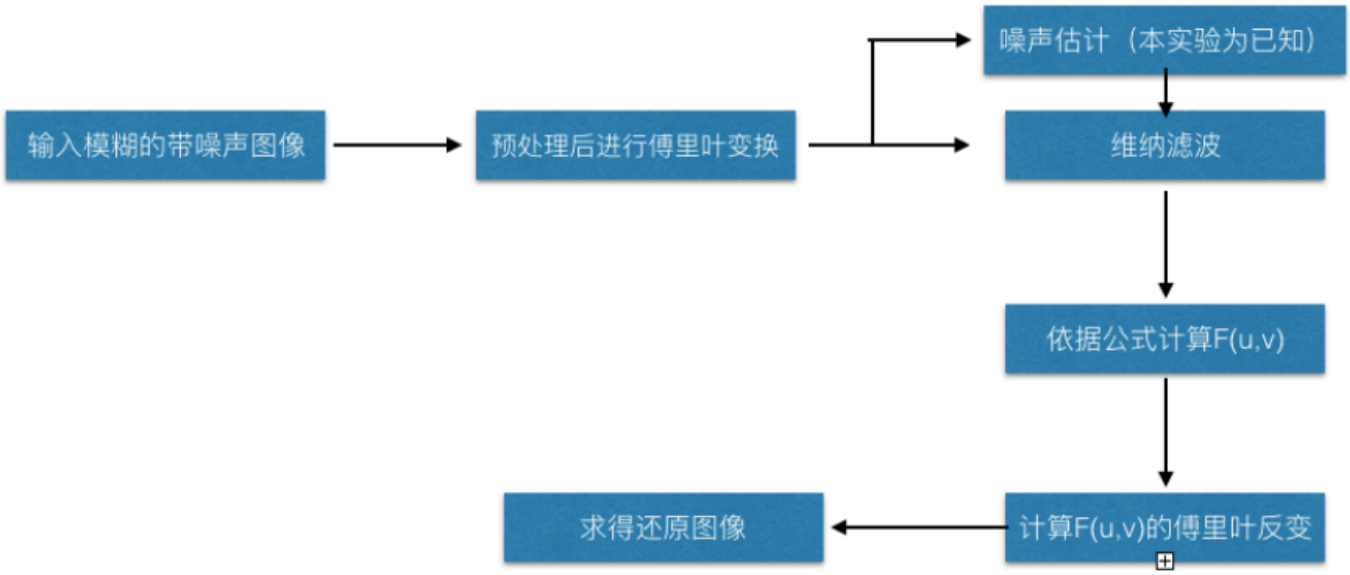
显然，当P_n(u,v) = 0时，维纳滤波复原法退化为逆滤波复原法。

当有噪声时，维纳滤波复原效果好于逆滤波。

算法步骤

1. 对退化图像g(x,y)进行二维离散傅里叶变换，得到G(u,v)
2. 计算系统冲激响应h(x,y)的二维离散傅立叶变换，得到H(u,v)，并将h(x,y)的尺寸拓展延伸至 g(x,y)的尺寸
3. 估算噪声的功率谱P*n(u,v)和输入图像的功率谱P*f(u,v)
4. 依公式计算F(u,v)
5. 计算F(u,v)的傅立叶反变换，求得还原图像f(x,y)

流程图



关于傅里叶变换

1-D 傅里叶变换：

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux}\, dx$$

它的逆变换：

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux}\, du$$

在此基础上很容易得到 2-D Fourier Transform:

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)}\, dx\, dy$$

以及逆变换：

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v) e^{j2\pi(ux+vy)}\, du\, dv$$

之前在学习复变函数与积分变换的时候，就觉得傅里叶变换是个很神奇的公式，因为他可以把在时域上看起来很复杂的东西投影到频域上，然后很贱的剥离一些成分，也就是这学期数字图像学习的各种滤波，感觉傅里叶变换其实相当于从另一个视角看待这个世界，很是让人震撼。

算法实现

```
from PIL import Image
import matplotlib.pyplot as ppt
import numpy

# 仿真运动模糊
def motion_process(len, size):
    sx, sy = size
    PSF = numpy.zeros((sy, sx))
    PSF[int(sy / 2):int(sy /2 + 1), int(sx / 2 - len / 2):int(sx / 2 + len / 2)] =
 1
    return PSF / PSF.sum() # 归一化亮度，之前忘记这一步，后来才发现

def make_blurred(input, PSF, eps):
    input_fft = numpy.fft.fft2(input) #图像傅里叶二维变换
    PSF_fft = numpy.fft.fft2(PSF) + eps
    blurred = numpy.fft.ifft2(input_fft * PSF_fft)
    blurred = numpy.abs(numpy.fft.fftshift(blurred))
    return blurred

# 维纳滤波
def wiener(input, PSF, eps):
    input_fft = numpy.fft.fft2(input)
    PSF_fft = numpy.fft.fft2(PSF) + ep s#噪声功率，这里是已知的，考虑epsilon
    result = numpy.fft.ifft2(input_fft / PSF_fft) #计算F(u,v)的傅里叶反变换
    result = numpy.abs(numpy.fft.fftshift(result))
    return result

image = Image.open('lena.jpg').convert('L')

# 显示原始图像
ppt.figure(1)
ppt.xlabel("Original Image")
ppt.gray()
ppt.imshow(image)

ppt.figure(2)
ppt.gray()

# 运动模糊处理
data = numpy.asarray(image.getdata()).reshape(image.size)
PSF = motion_process(30, data.shape)
blurred = numpy.abs(make_blurred(data, PSF, 1e-3))

# 显示运动模糊后的图像
ppt.subplot(221)
ppt.xlabel("Motion blurred")
```

```
ppt.imshow(blurred)

# 图像恢复
result = wiener(blurred, PSF, 1e-3)
ppt.subplot(222)
ppt.xlabel("wiener deblurred")
ppt.imshow(result)

# 添加噪声
blurred += 0.1 * blurred.std() * numpy.random.standard_normal(blurred.shape)

# 显示添加噪声且运动模糊的图像
ppt.subplot(223)
ppt.xlabel("motion & noisy blurred")
ppt.imshow(blurred)

# 图像恢复
result = wiener(blurred, PSF, 0.1 + 1e-3)
ppt.subplot(224)
ppt.xlabel("wiener deblurred")
ppt.imshow(result)

ppt.show()
```

结果

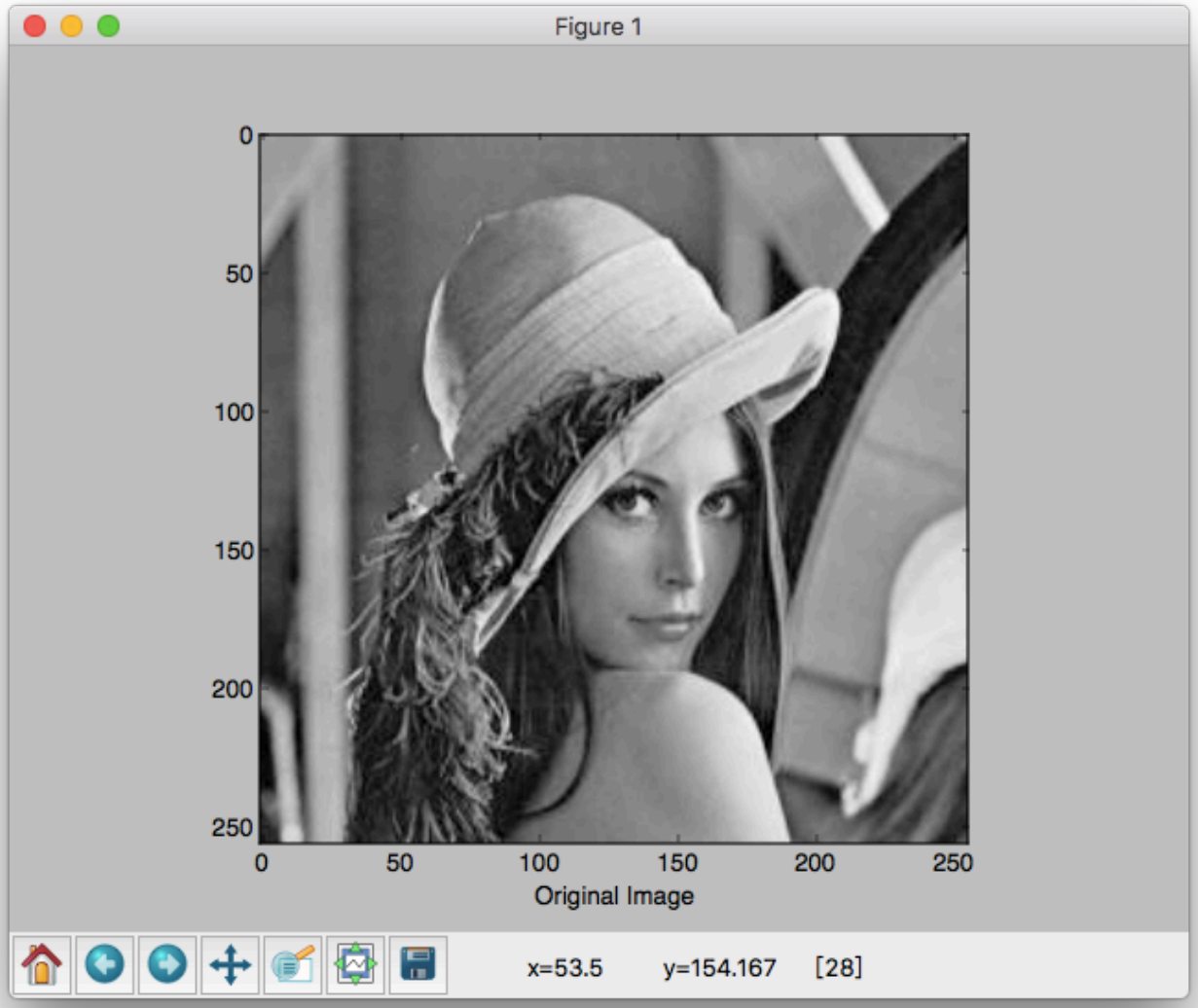
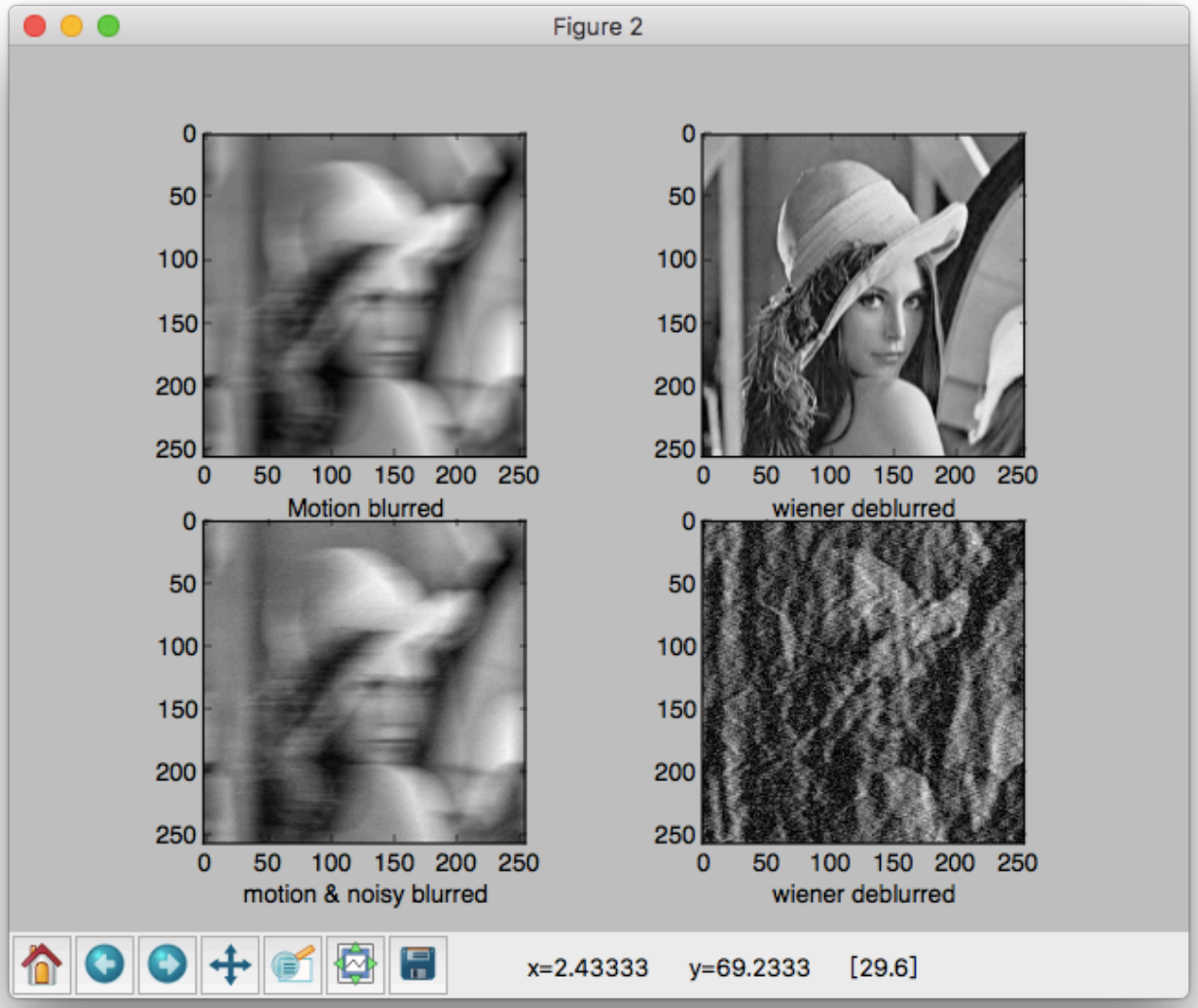


Figure 1
0
50
100
150
200
250
0
50
100
150
200
250
Original Image
x=53.5
y=154.167
[28]



可以发现，仅有运动模糊，没有噪声时，是可以完美恢复的，但是当加入噪声后，即使考虑了噪声参数，也是有一定影响，但总体还可看清轮廓。

不足

这个实验我虽然是自己手工实现了维纳滤波，但是傅里叶变换我却是调用了numpy，其实，我前前后后花了很多时间来手工实现傅里叶变换，一维的变换可以实现，但是二维快速傅里叶变换直到最后仍有问题，最终只得放弃，改用numpy的傅里叶变换。 下面是我手工实现傅里叶变换的过程和探索经历，记录下来后续反思：

首先我自己手写了一个一维的傅里叶变换

```
from cmath import exp, pi

def fft(x):
    N = len(x)
    if N <= 1: return x
    even = fft(x[0::2])
    odd =  fft(x[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]
```

我比较了他和numpy里fft产生的结果，是一致的，然后我就在此基础上利用矩阵写了个二维傅里叶变换:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

#get the dft_matrix
def dft_matrix(N):
    i,j = np.meshgrid(np.arange(N),np.arange(N))
    omega = np.exp(-2j*np.pi/N)
    w = np.power(omega,i*j)
    return w
def dft2d(image,flags):
    h,w = image.shape[:2]
    image_shift = np.zeros((h,w),np.uint8)
    output = np.zeros((h,w),np.complex)
    # for x in xrange(h):
    #   for y in xrange(w):
    #       image_shift[x,y] = image[x,y]*(-1)**(x+y)
    output = dft_matrix(h).dot(image).dot(dft_matrix(w))
    return output
```

但是生成的图像惨不忍睹。。。我就不放图了，复原的结果几乎都成了一条直线。

最后我选择参考numpy的源码，我几乎又把numpy重新实现了一遍，如下：

```
from PIL import Image
import matplotlib.pyplot as ppt
import numpy
from cmath import exp, pi

# Fourier Transforms

_fft_cache = {}

def cfftb(*args, **kwargs): # real signature unknown
```

```
    pass

def cfftf(*args, **kwargs): # real signature unknown
    pass

def cffti(*args, **kwargs): # real signature unknown
    pass
#


# 一维傅里叶变换计算函数，核心部分
def _raw_cal(x, n=None, axis=-1, init_function=cffti, work_function=cfftf, fft_cac
he=_fft_cache):
    x = numpy.asarray(x)
    if n is None:
        n = x.shape[axis]

    if n < 1:
        raise ValueError("Invalid number of FFT data points (%d" % n)

    try:
        wsave = fft_cache.setdefault(n, []).pop()
    except (IndexError):
        wsave = init_function(n)

    if x.shape[axis] != n:
        s = list(x.shape)
        if s[axis] > n:
            index = [slice(None)]*len(s)
            index[axis] = slice(0, n)
            x = x[index]
        else:
            index = [slice(None)]*len(s)
            index[axis] = slice(0, s[axis])
            s[axis] = n
            z = numpy.zeros(s, x.dtype.char)
            z[index] = x
            x = z

    if axis != -1:
        x = numpy.swapaxes(x, axis, -1)
    r = work_function(x, wsave)
    if axis != -1:
        r = numpy.swapaxes(r, axis, -1)

    fft_cache[n].append(wsave)

    return r
```

```
# 一维傅里叶变换调用
def fft(x, n=None, axis=-1, norm=None):
    x = numpy.asarray(x).astype(complex, copy=False)
    if n is None:
        n = x.shape[axis]
    output = _raw_cal(x, n, axis, cffti, cfftf, _fft_cache)

    if _unitary(norm):
        output *= 1 / numpy.sqrt(n)
    return output


def Reshape(x, s=None, axes=None, invreal=0):
    if s is None:
        shapeless = 1
        if axes is None:
            s = list(x.shape)
        else:
            s = numpy.take(x.shape, axes)
    else:
        shapeless = 0
    s = list(s)
    if axes is None:
        axes = list(range(-len(s), 0))
    if len(s) != len((axes)):
        raise ValueError("Shape and axes have different lengths.")
    if invreal and shapeless:
        s[-1] = (x.shape[axes[-1]]-1) * 2
    return s, axes


# 傅里叶变换预处理，调用一维的傅里叶变换
def _raw_fft(x, s=None, axes=None, function=fft, norm=None):
    x = numpy.asarray(x)
    s, axes = Reshape(x, s, axes)
    itl = list(range(len(axes)))
    itl.reverse()
    for ii in itl:
        x = function(x, n=s[ii], axis=axes[ii], norm=norm)
    return x

# 二维傅里叶变换驱动函数
def FFT2(x, s=None, axes=(-2, -1), norm=None):
    return _raw_fft(x, s, axes, fft, norm)


def _unitary(norm):
```

```
   if norm not in (None, "ortho"):
       raise ValueError("Invalid norm value %s" % norm)
   return norm is not None

# 逆傅里叶变换
def ifft(x, n=None, axis=-1, norm=None):
    x = numpy.array(x, copy=True, dtype=complex)
    if n is None:
        n = x.shape[axis]
    unitary = _unitary(norm)
    output = _raw_cal(x, n, axis, cffti, cfftb, _fft_cache)
    return output * (1/numpy.sqrt(n) if unitary else n)

# 二维逆傅里叶变换驱动函数
def iFFT2(x, s=None, axes=(-2, -1), norm=None):
    return _raw_fft(x, s, ifft, norm)



def motion_process(len, size):
    sx, sy = size
    PSF = numpy.zeros((sy, sx))
    PSF[int(sy / 2):int(sy /2 + 1), int(sx / 2 - len / 2):int(sx / 2 + len / 2)] =
 1
    return PSF / PSF.sum()

def make_blurred(input, PSF, eps):
    input_fft = FFT2(input)
    PSF_fft = FFT2(PSF) + eps
    blurred = iFFT2(input_fft * PSF_fft)
    blurred = numpy.abs(numpy.fft.fftshift(blurred))
    return blurred

def wiener(input, PSF, eps):
    input_fft = FFT2(input)
    PSF_fft = FFT2(PSF) + eps
    result = iFFT2(input_fft / PSF_fft)
    result = numpy.abs(numpy.fft.fftshift(result))
    return result


# main
image = Image.open('lena.jpg').convert('L')
ppt.figure(1)
ppt.xlabel("Original Image")
ppt.gray()
ppt.imshow(image)
```

```
ppt.figure(2)
ppt.gray()
data = numpy.asarray(image.getdata()).reshape(image.size)
PSF = motion_process(30, data.shape)
blurred = numpy.abs(make_blurred(data, PSF, 1e-3))

ppt.subplot(221)
ppt.xlabel("Motion blurred")
ppt.imshow(blurred)

result = wiener(blurred, PSF, 1e-3)
ppt.subplot(222)
ppt.xlabel("wiener deblurred")
ppt.imshow(result)

blurred += 0.1 * blurred.std() * numpy.random.standard_normal(blurred.shape)

ppt.subplot(223)
ppt.xlabel("motion & noisy blurred")
ppt.imshow(blurred)

result = wiener(blurred, PSF, 0.1 + 1e-3)
ppt.subplot(224)
ppt.xlabel("wiener deblurred")
ppt.imshow(result)

ppt.show()
```

但是运行报错如下：

我google了这个错误，在网上找到了一些解决办法试验之后，至今都无法解决。本学期选了5门选修课程，很多考试在即，最后没有解决这个bug，是个不足。