中山大學
SUN YAT-SEN UNIVERSITY

《数字图像处理》项目报告

Digital Image Processing Project Report

项目章节：　　第四章

项目编号：　　04-01, 04-02, 04-03, 04-04, 04-05,Addition

姓名：　　刘昕、许皓、郭达雅

学号：　　14348085、14348146、14348024

摘要：

使用 Python3 完成图像傅立叶变换并且使其中心化（04-01），将滤波从空间域转换到频域，通过直流分量求的图像的均值（04-02），将空间域的卷积转换到频域的乘积，在频域利用高斯低通滤波器对图像进行模糊化（04-03），从原图像中减去模糊的部分，得到锐化的图像（04-04），最后，利用零延拓和相关函数，检测图像的相关性，并找出相关性最强的位置（04-05），最后探究谱平面随图片旋转的性质（Addition）。

时间：　2017 年 4 月 8 日

目录

PROJECT 04-01 Two-Dimensional Fast Fourier Transform

实验要求

The purpose of this project is to develop a 2-D FFT program "package" that will be used in several other projects that follow. Your implementation must have the capabilities to:

(a) Multiply the input image by $(-1)^{x+y}$ to center the transform for filtering.

(b) Multiply the resulting (complex) array by a real function (in the sense that the the real coefficients multiply both the real and imaginary parts of the transforms). Recall that multiplication of two images is done on pairs of corresponding elements.

(c) Compute the inverse Fourier transform.

(d) Multiply the result by $(-1)^{x+y}$

(e) Compute the spectrum. and take the real part.

Basically, this project implements Fig. 4.5. If you are using MATLAB, then your Fourier transform program will not be limited to images whose size are integer powers of 2. If you are implementing the program yourself, then the FFT routine you are using may be limited to integer powers of 2. In this case, you may need to zoom or shrink an image to the proper size by using the program you developed in Project 02-04.

An approximation: To simplify this and the following projects (with the exception of Project 04-05), you may ignore image padding (Section 4.6.3). Although your results will not be strictly correct, significant simplifications will be gained not only in image sizes, but also in the need for cropping the final result. The principles will not be affected by this approximation.

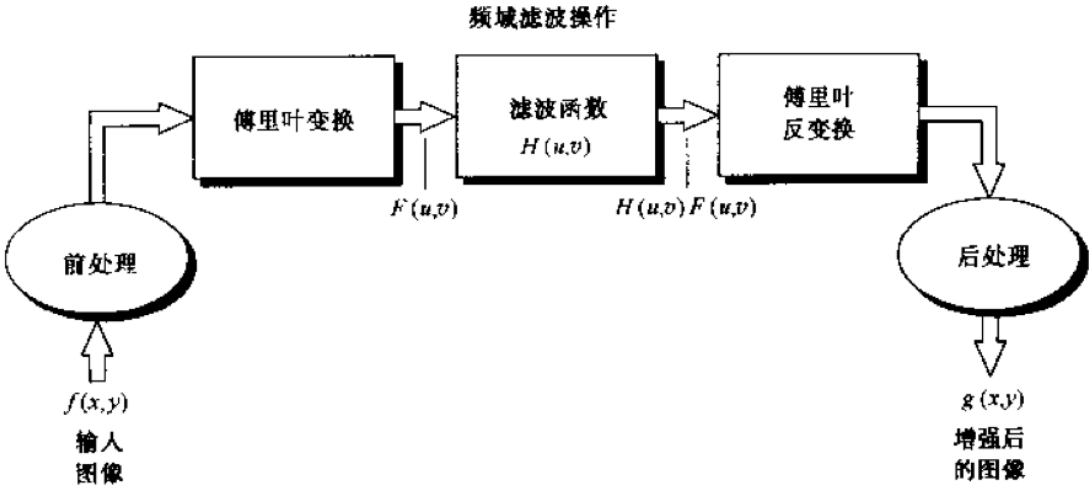

图 4.5　频域滤波的基本步骤

图 1Fig.4.5(a)

技术讨论

(1) 傅立叶变换能使我们从空间域（或空间域）与频率域两个不同的角度来看待信号或图象的问题。有时在空间域无法解决的问题，在频域却是显 而易见的。

(2) 傅里叶分析中最重要的结论就是几乎 “ 所有 ” 的函数 (信号) 都可以表示为 (分解成) 简单的 (加权) 正弦波和余弦波之和。

(3) 定义二维离散情形的傅里叶变换 (DFT) 公式 : $f(x,y)$ 为离散函数 , 其中 $x = 0,1,\dots,M-1, y = 0,1,\dots,N-1,$.

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

对应的傅立叶反变换也总是存在的

$$f(x,y) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} F(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

(4) 为了把变换后的中心移到图像的中心 $(\frac{M}{2},\frac{N}{2})$, 通常在变换之前都要在函数上乘以 $(-1)^{x+y}$. 这是由于傅里叶变换的平移性质决定的。

$$f(x,y)(-1)^{x+y} \Leftrightarrow F(u-\frac{M}{2},v-\frac{N}{2})$$

$$F(u,v)(-1)^{u+v} \Leftrightarrow f(x-\frac{M}{2},y-\frac{N}{2})$$

(5) 两个 $M\times N$的离散函数 $f(x,y)$ 和 $h(x,y)$ 的卷积定义为

$$f(x,y) * h(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)h(x-m,y-n)$$

空间域滤波的本质上就是用选好的掩模 (mask), 经过一定的处理 , 与给定的图像作卷积

(6) 卷积定理：在空间域的卷积就是在频域的乘积，在空间域的乘积就是在频域的卷积

$$f(x,y) * h(x,y) \Leftrightarrow F(u,v)H(u,v)$$

$$f(x,y)h(x,y) \Leftrightarrow F(u,v) * H(u,v)$$

其中

$$f(x,y) \Leftrightarrow F(u,v)$$

$$h(x,y) \Leftrightarrow H(u,v)$$

结果讨论

(1) 对图像的部分处理实际上是要在空间域进行卷积操作，但是卷积不容易计算，利用傅立叶变换，将空间域的图像转换到频率域，利用卷积定理，可以将复杂的卷积操作变成简单的乘积操作，大大简化计算

结果

略

附录

```
#!/usr/bin/env python3
#coding:utf-8

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
```

```
def my_FFT(image):
    height, width = image.shape
    FFT_image = np.empty((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            coff = -1 if (i+j) & 1 else 1
            FFT_image[i, j] = coff * image[i, j]
    return np.fft.fft2(FFT_image)


def my_iFFT(FFT_image):
    image_back = np.fft.ifft2(FFT_image)
    height, width = image_back.shape
    for i in range(height):
        for j in range(0 if i & 1 else 1, width, 2):
            image_back[i, j] = -image_back[i, j]
    return image_back
```

PROJECT 04-02 Fourier Spectrum and Average Value

实验要求

(a) Download Fig. 4.18(a) and compute its (centered) Fourier spectrum.

(b) Display the spectrum.

(c) Use your result in (a) to compute the average value of the image.

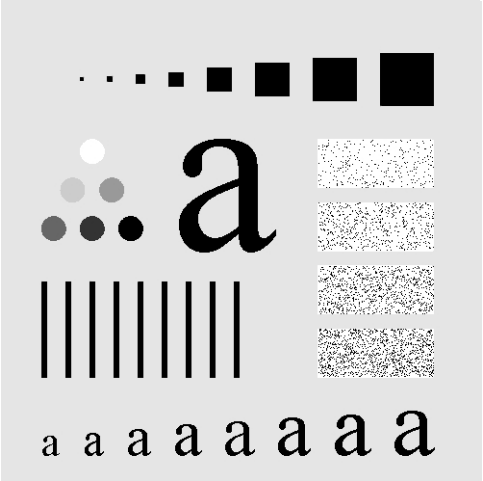

图 2Fig.4.11(a)

技术讨论

(1) 我们把量 $|F(u)| = [R^2(u) + I^2(u)]^{\frac{1}{2}}$ 称为傅里叶变换的幅度 (Magnitude) 或者谱 (Spectrum)。谱可以表示原函 数 (或图像) 对某一频谱分量的贡献。

(2) 我们可以通过频域中直流分量（频率等于 0 的部分）求的图像在空间域中的均值，表示图像较为平滑没有太大变化的部分。

$$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$$

结果讨论

(1) 利用 04-01 实现的 FFT，很容易求的图像的谱

(2) 但是由于图像由傅立叶变换变换到频率域，有实部和虚部。范围超出了图像灰度级的 0~255，所以我们为了显示图像的谱，需要对谱进行处理。在本实验中，

先对每一项取模，然后取对数，目的是使它们限制在 0~255 之间，并且容易观察出谱的分布

(3) 利用直流分量，很容易求得图像的均值。但是需要注意的是，因为谱进行了中心化，所以直流分量并不是$F(0,0)$，而是$F(\frac{M}{2},\frac{N}{2})$，同时还需要乘系数$\frac{1}{MN}$进行调整（系数根据 Python 实现的傅立叶变换所确定）。最后求得结果为 207

结果

(a)

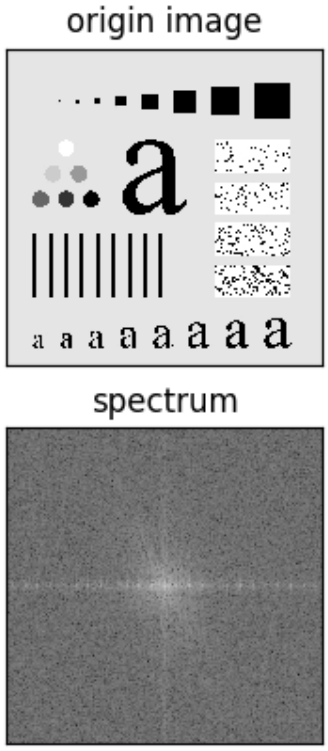

图 3Project04-02 结果

(b)图像的均值为 207

附录

```
#/usr/bin/env python3
#coding:utf-8
```

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

def my_normalize(image):
    scaled_image = np.frompyfunc(lambda x: max(0, min(x, 255)), 1, 1)(image).astype(np.uint8)
    return scaled_image

def my_FFT(image):
    height, width = image.shape
    FFT_image = np.empty((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            coff = -1 if (i+j) & 1 else 1
            FFT_image[i, j] = coff * image[i, j]
    return np.fft.fft2(FFT_image)

img = np.array(Image.open('../images/images_chapter_04/Fig4.11(a).jpg').convert('L'))
fft_img = my_FFT(img)
ax = plt.subplot(2,1,1)
ax.set_xticks([])
ax.set_yticks([])
plt.title('origin image')
plt.imshow(img, cmap='gray')
ax = plt.subplot(2,1,2)
ax.set_xticks([])
ax.set_yticks([])
plt.title('spectrum')
```

```
plt.imshow(np.log(np.abs(fft_img)), cmap='gray')
plt.show()
height, weight = fft_img.shape
print('The average value is', fft_img[height // 2, weight // 2].real / height / weight)
```

PROJECT 04-03 Lowpass Filtering

实验要求

(a) Implement the Gaussian lowpass filter in Eq. (4.3-7). You must be able to specify the size, M x N, of the resulting 2D function. In addition, you must be able to specify where the 2D location of the center of the Gaussian function.

(b) Download Fig. 4.11(a) [this image is the same as Fig. 4.18(a)] and lowpass filter it to obtain Fig. 4.18(c).

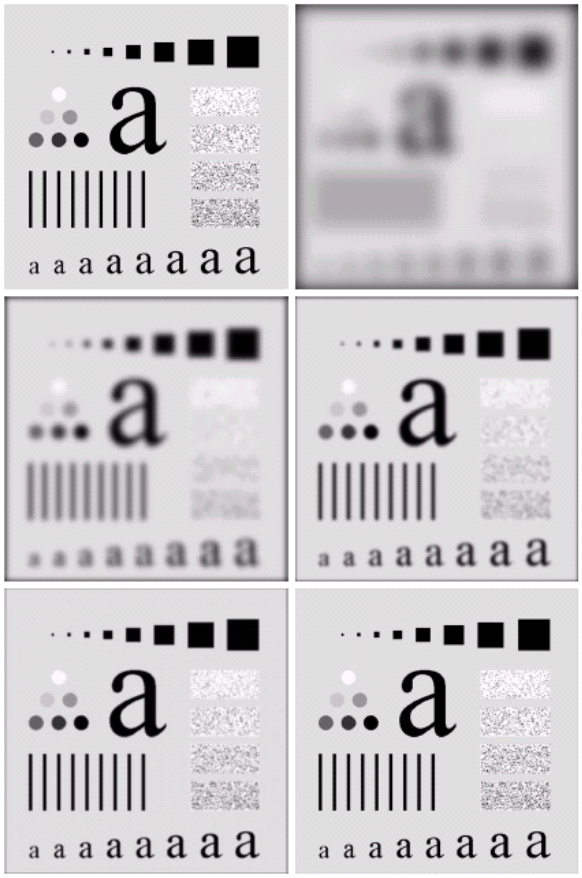

FIGURE 4.18 (a) Original image. (b)–(f) Results of filtering with Gaussian lowpass filters with cutoff frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Figs. 4.12 and 4.15. a b c d e f

图 4Fig.4.18

技术讨论

(1) 高斯低通滤波器 (Gaussian Lowpass Filter)是一种常用的低通滤波器，其形式简单，在空间域和频率域都具有相似的表达。虽然不如 2 阶 BLPF 的模糊效果好.

高斯滤波器完全保证没有振铃，这在很多人为误差无法接受的应用中(如医学图像)是非常重要的。

(2) 利用傅立叶变换，在空间域复杂的卷积很容易可以在频率域利用乘积实现。先将原图像进行傅立叶变换，并且中心化，然后乘以高斯低通滤波器的傅立叶变换，最后得到模糊化的图像。

结果讨论

(1) 高斯变换的确可以起到模糊化的效果，并且可以发现，没有理想低通滤波器和巴特沃斯滤波器的振铃现象

结果

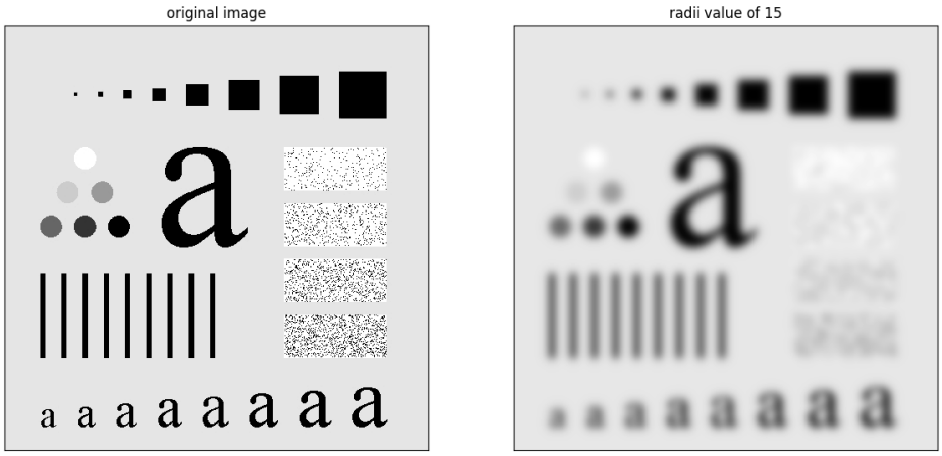

图 5Project04-03 结果

附录

```
#/usr/bin/env python3
#coding:utf-8

from PIL import Image
```

```
import matplotlib.pyplot as plt
import numpy as np


def my_FFT(image):
    height, width = image.shape
    FFT_image = np.empty((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            coff = -1 if (i+j) & 1 else 1
            FFT_image[i, j] = coff * image[i, j]
    return np.fft.fft2(FFT_image)

def my_iFFT(FFT_image):
    image_back = np.fft.ifft2(FFT_image).real
    height, width = image_back.shape
    for i in range(height):
        for j in range(0 if i & 1 else 1, width, 2):
            image_back[i, j] = -image_back[i, j]
    return image_back

def my_Gaussian_Lowpass(image, d_0):
    fft_img = my_FFT(image)
    height, width = image.shape
    u0 = 0
    v0 = 0
    for i in range(height):
        for j in range(width):
            if fft_img[i,j] > fft_img[u0, v0]:
                u0 = i
```

```
            v0 = j
    d2 = lambda u, v: (u - height // 2) * (u - height // 2) + (v - width // 2) * (v - width //
2)
    H = np.empty((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            H[i, j] = np.exp(-d2(i, j) / 2 / d_0 / d_0)
    lowpass_image = my_iFFT(H * fft_img)
    return lowpass_image

img = np.array(Image.open('../images/images_chapter_04/Fig4.11(a).jpg').convert('L'))
ax = plt.subplot(1,2,1)
ax.set_xticks([])
ax.set_yticks([])
plt.title('original image')
plt.imshow(img, cmap='gray')
ax = plt.subplot(1,2,2)
ax.set_xticks([])
ax.set_yticks([])
plt.title('radii value of 15')
plt.imshow(my_Gaussian_Lowpass(img, 15), cmap='gray')
plt.show()
```

PROJECT 04-04 Highpass Filtering Using a Lowpass Image

实验要求

(a) Subtract your image in Project 04-03(b) from the original to obtain a sharpened image, as in Eq. (4.4-14). You will note that the resulting image does not resemble the Gaussian highpass results in Fig. 4.26. Explain why this is so.

(b) Adjust the variance of your Gaussian lowpass filter until the result obtained by image subtraction looks similar to Fig. 4.26(c). Explain your result.

钝化模板简单地由从一幅图像减去其自身模糊图像而生成的锐化图像构成。采用频域技术，这意味着从图像自身减去低通滤波后的图像而得到高通滤波的图像。即：

$$f_{hp}(x,y) = f(x,y) - f_{lp}(x,y) \qquad (4.4.14)$$

图 6 Eq. (4.4-14)

a b c

FIGURE 4.26 Results of highpass filtering the image of Fig. 4.11(a) using a GHPF of order 2 with $D_0 = 15$, 30, and 80, respectively. Compare with Figs. 4.24 and 4.25.

图 7 Fig4.26

技术讨论

(1) Eq 4.4-14 给出的是空间域的公式，含意是锐化的图像＝原图像－经过低通高斯滤波得到的图像。

结果讨论

(1) 高斯变换的确可以起到模糊化的效果，并且可以发现，没有理想低通滤波器和

巴特沃斯滤波器的振铃现象

(2)但是由 Eq 4.4-14 得到的$D_0 = 15$的图像和 Fig4.26 中的图像比，实验获得的图片细节更加少。因为 Fig4.26 中使用了 2 次高斯高通滤波，即

$$H(u,v)H(u,v)F(u,v) = \left(1 - e^{-\frac{D^2(u,v)}{2D_0^2}}\right)^2 F(u,v) \approx \left(1 - e^{-\frac{D^2(u,v)}{D_0^2}}\right)F(u,v)$$

Eq 4.4-14 得到的等价的滤波器为

$$G(u,v) = H(u,v) = \left(1 - e^{-\frac{D^2(u,v)}{2D_0^2}}\right)$$

对比可知，实验中使用的D_0和 Fig4.26 的D_0'有如下关系

$$D_0 = \frac{D_0'}{\sqrt{2}}$$

(3)经过计算和测试，当方差$D_0 = \frac{80}{\sqrt{2}} = 56.57$左右时，效果同 Fig4.26 相似

结果

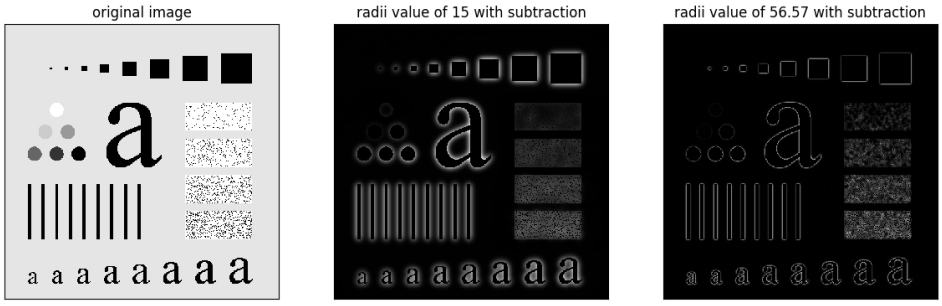

图 8Project04-04 结果

附录

```
#/usr/bin/env python3
#coding:utf-8
```

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

def my_normalize(image):
    scaled_image = np.frompyfunc(lambda x: max(0, min(x, 255)), 1, 1)(image).astype(np.uint8)
    return scaled_image

def my_FFT(image):
    height, width = image.shape
    FFT_image = np.empty((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            coff = -1 if (i+j) & 1 else 1
            FFT_image[i, j] = coff * image[i, j]
    return np.fft.fft2(FFT_image)

def my_iFFT(FFT_image):
    image_back = np.fft.ifft2(FFT_image).real
    height, width = image_back.shape
    for i in range(height):
        for j in range(0 if i & 1 else 1, width, 2):
            image_back[i, j] = -image_back[i, j]
    return image_back

def my_Gaussian_Lowpass(image, d_0):
    fft_img = my_FFT(image)
    height, width = image.shape
```

```
    d2 = lambda u, v: (u - height // 2) * (u - height // 2) + (v - width // 2) * (v - width //
2)
    H = np.empty((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            H[i, j] = np.exp(-d2(i, j) / 2 / d_0 / d_0)
    lowpass_image = my_iFFT(H * fft_img)
    return lowpass_image

def my_Gaussian_Highpass(image, d_0):
    lowpass_img = my_Gaussian_Lowpass(image, d_0)
    return image - my_normalize(lowpass_img).astype(np.float64)

img = np.array(Image.open('../images/images_chapter_04/Fig4.11(a).jpg').convert('L'))
img1 = my_Gaussian_Highpass(img, 15)
d_0 = 56.57
img2 = my_Gaussian_Highpass(img, d_0)
ax = plt.subplot(1,3,1)
ax.set_xticks([])
ax.set_yticks([])
plt.title('original image')
plt.imshow(img, cmap='gray')
ax = plt.subplot(1,3,2)
ax.set_xticks([])
ax.set_yticks([])
plt.title('radii value of 15 with subtraction')
plt.imshow(my_normalize(img1), cmap='gray')
ax = plt.subplot(1,3,3)
ax.set_xticks([])
ax.set_yticks([])
```

```
plt.title('radii value of %.2f with subtraction' % d_0)
plt.imshow(my_normalize(img2), cmap='gray')
plt.show()
```

PROJECT 04-05 Correlation in the Frequency Domain

实验要求

(a) Download Figs. 4.41(a) and (b) and duplicate Example 4.11 to obtain Fig. 4.41(e). Give the (x,y) coordinates of the location of the maximum value in the 2D correlation function. There is no need to plot the profile in Fig. 4.41(f).

图 10 Fig4.41(a)

图 11 Fig4.41(b)

技术讨论

(1) 周期问题是傅立叶变换中可能出现的问题。当周期不当时，会出现重叠的情况。处理的方法是对要处理的函数作零延拓(Padding).假设函数$f(x)$和$h(x)$分别有A个点和B个点组成,对两个函数同时添加零,使它们具有相同的周期,用P表示

$$f_e = \begin{cases} f(x), 0 \le x \le A-1 \\ 0, A \le x \le P \end{cases}$$

$$h_e = \begin{cases} h(x), 0 \le x \le B-1 \\ 0, B \le x \le P \end{cases}$$

只要$P \geq A + B - 1$,将$f_e(x)$和$h_e(x)$作周期化处理

(2) 在频率域计算卷积的正确步骤

(1) 对两个函数作零延拓至适当的长度;

(2) 做两个序列的傅里叶变换(长度和延拓后的一致);

(3) 将两个序列的傅里叶变换相乘;

(4) 计算乘积的傅里叶反变换.

(3) 两个二维函数$f(x,y)$和$h(x,y)$的相关函数定义为:

$$g(x,y) = f(x,y) \circ h(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f^*(m,n) h(x+m, y+n)$$

f^*表示f的复共轭.注意相关函数和卷积有非常类似的结构,因此

$$f(x,y) \circ h(x,y) \Leftrightarrow F^*(u,v) H(u,v)$$

$$f^*(x,y) \circ h(x,y) \Leftrightarrow F(u,v) H(u,v)$$

(4) 假设$f(x,y)$是一幅包含物体或者区域的图像,如果想知道中是否包含有感兴趣的物体或区域,就让$h(x,y)$作为那个区域或物体(模板),然后求两者的相关函数.如果存在匹配即f中有感兴趣的物体或区域,则相关函数会在h和f对应的位置达到最大值.

结果讨论

⑴通过零延拓、傅立叶变换、求共轭、乘积、傅立叶反变换，最终得到的结果和 Fig4.41 相同。可以很明显看出，在“UTK”的“T”处，有较强的灰度值，这里就是最大相似的地方

⑵通过上述实验，说明利用傅立叶变换可以很容易找到图像的最大相似地方，但是缺点是对图像的大小有要求，如果模板的大小和图像中相似的部分大小不相同，该方法就不奏效

结果

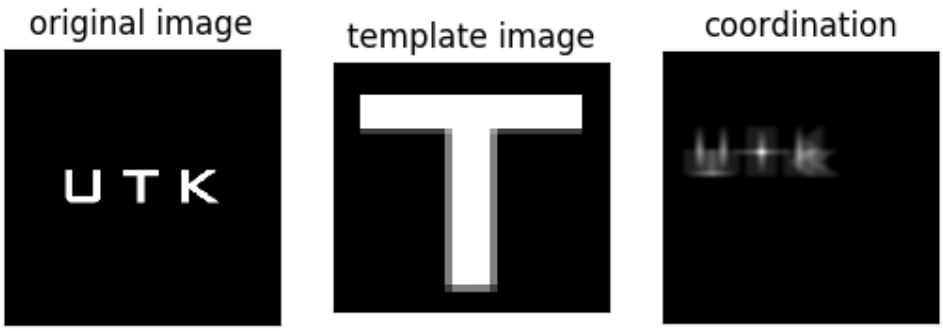

图 12Project04-05 结果

附录

```
#/usr/bin/env python3
#coding:utf-8

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

def my_FFT(image):
    height, width = image.shape
    FFT_image = np.empty((height, width), dtype=np.float64)
```

```
    for i in range(height):
        for j in range(width):
            coff = -1 if (i+j) & 1 else 1
            FFT_image[i, j] = coff * image[i, j]
    return np.fft.fft2(FFT_image)

def my_iFFT(FFT_image):
    image_back = np.fft.ifft2(FFT_image).real
    height, width = image_back.shape
    for i in range(height):
        for j in range(0 if i & 1 else 1, width, 2):
            image_back[i, j] = -image_back[i, j]
    return image_back

def my_padding(image1, image2):
    height1, width1 = image1.shape
    height2, width2 = image2.shape
    height = height1 + height2 - 1
    width = width1 + width2 - 1
    padded_image1 = np.zeros((height, width), dtype=np.uint8)
    padded_image2 = np.zeros((height, width), dtype=np.uint8)
    for i in range(height1):
        for j in range(width1):
            padded_image1[i,j] = image1[i,j]
    for i in range(height2):
        for j in range(width2):
            padded_image2[i,j] = image2[i,j]
    return padded_image1, padded_image2

def my_coordination(image1, image2):
```

```
    fft_image1 = my_FFT(image1)
    fft_image2_conj = my_FFT(image2).conj()
    coordinating_image = fft_image1 * fft_image2_conj
    return my_iFFT(coordinating_image)

original_img = np.array(Image.open('../images/images_chapter_04/Fig4.41(a).jpg').convert('L'))
template_img = np.array(Image.open('../images/images_chapter_04/Fig4.41(b).jpg').convert('L'))
padded_original_img, padded_template_img = my_padding(original_img, template_img)
coordinating_img = my_coordination(padded_original_img, padded_template_img)
position = np.argmax(coordinating_img)
height, width = coordinating_img.shape
x = position // height
y = position % width
print(x, y)
ax = plt.subplot(1,3,1)
ax.set_xticks([])
ax.set_yticks([])
plt.title('original image')
plt.imshow(original_img, cmap='gray')
ax = plt.subplot(1,3,2)
ax.set_xticks([])
ax.set_yticks([])
plt.title('template image')
plt.imshow(template_img, cmap='gray')
ax = plt.subplot(1,3,3)
ax.set_xticks([])
ax.set_yticks([])
```

```
plt.title('coordination')
plt.imshow(coordinating_img, cmap='gray')
plt.show()
```

PROJECT Addition The spectral plane rotates with the image

实验要求

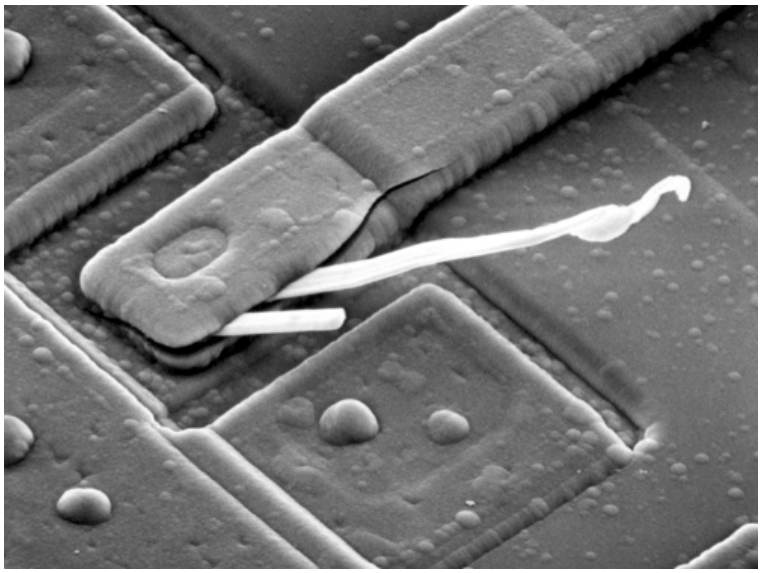

图 12 Fig4.04(a)

技术讨论

(1) 利用 Python 库函数，实现图像的旋转，然后利用 04-01 的函数，很容易求得旋转后图像的谱平面，利用观察可以找到谱平面随图像旋转的性质

结果讨论

(1) 根据图像可知，图像旋转 90°时，谱平面旋转 90°；图像旋转 180°，谱平面旋转 180°，图像旋转 270°，谱平面旋转 270°

(2) 如果旋转的角度θ不是 90°的整数倍，那么图像会填充黑色（灰度值为 0），导致图像灰度值分布变化较大，导致谱平面没有很明显的相关性。

结果

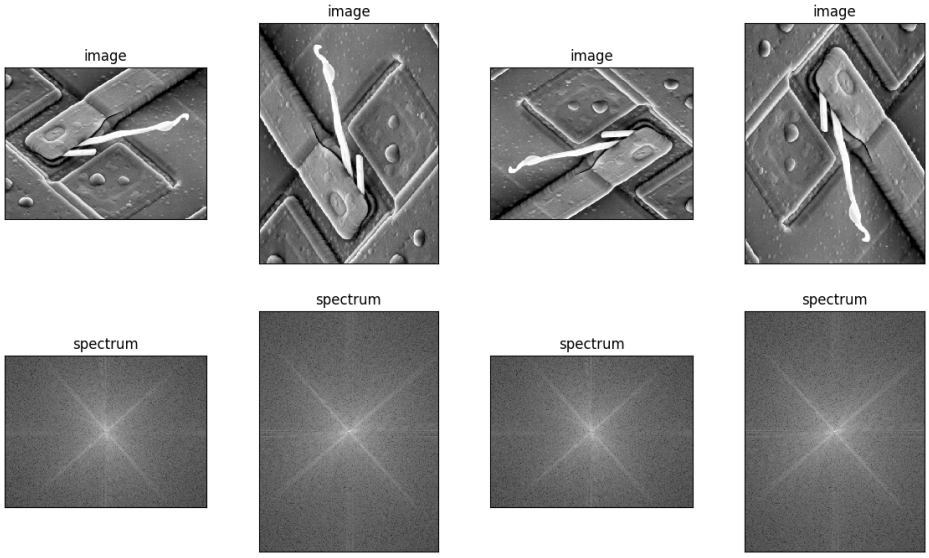

图 13Addition 结果

附录

```
#!/usr/bin/env python3
#coding:utf-8

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

def my_FFT(image):
    height, width = image.shape
    FFT_image = np.empty((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            coff = -1 if (i+j) & 1 else 1
```

```
            FFT_image[i, j] = coff * image[i, j]
    return np.fft.fft2(FFT_image)


def my_iFFT(FFT_image):
    image_back = np.fft.ifft2(FFT_image)
    height, width = image_back.shape
    for i in range(height):
        for j in range(0 if i & 1 else 1, width, 2):
            image_back[i, j] = -image_back[i, j]
    return image_back


imgs = []
image = Image.open('../images/images_chapter_04/Fig4.04(a).jpg')
imgs.append(np.array(image.convert('L')))
imgs.append(np.array(image.transpose(Image.ROTATE_90).convert('L')))
imgs.append(np.array(image.transpose(Image.ROTATE_180).convert('L')))
imgs.append(np.array(image.transpose(Image.ROTATE_270).convert('L')))
fft_imgs = []
for i in range(len(imgs)):
    fft_imgs.append(my_FFT(imgs[i]))
    ax = plt.subplot(2,len(imgs),1 + i)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.title('image')
    plt.imshow(imgs[i], cmap='gray')
    ax = plt.subplot(2,len(imgs),1 + i + len(imgs))
    ax.set_xticks([])
    ax.set_yticks([])
    plt.title('spectrum')
    plt.imshow(np.log(np.abs(fft_imgs[i])), cmap='gray')
```

```
plt.show()
```