数字图像处理实验报告

姓　　名：张子宁
学　　号：U202215126
班　　级：人工智能 2204
任课教师：肖阳
成　　绩：
时　　间：2024.9.24
单　　位：人工智能与自动化学院

目录

1 实验任务

1. 结合授课内容和自身的理解，请尝试阐述傅里叶变换与傅里叶系数的物理含义，可以上网查阅相关的资料。

2. 编写程序（建议 Matlab）对图 1（自行转换为灰度图）展开（1）顺时针旋转 30 度；（2）基于最近邻和双线性插值将图像分别放大 2 倍和 4 倍，并形成实验报告。

3. 编写程序（建议 Matlab）对以上图像（自行转换为灰度图）展开傅里叶变换，提取傅里叶变换图像（将频率原点移至图像中心），并形成实验报告。

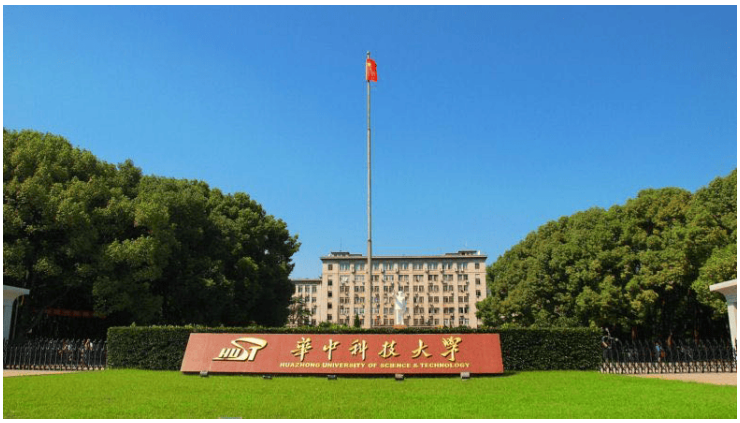

图 1: 任务图片

2 实验内容

2.1 实验内容一

2.1.1 傅里叶变换的物理含义

傅里叶变换是对非周期信号的一种分析方法，将时域信号转换为频域信号。对于信号 $f(t)$，其傅里叶变换 $F(\omega)$ 表示为：

$$F(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-j\omega t}dt$$

物理意义：

1. **频谱分析**：傅里叶变换提供了信号的频谱，可以直观地看出各个频率成分的强度与相位关系。这对于理解信号的频域特性至关重要。

2. **时域与频域的对应**：通过傅里叶变换，可以实现时域与频域之间的转换，很多物理现象在频域中更易于分析。例如，滤波、卷积等操作在频域中计算更为简便。

3. **信号的分解**：傅里叶变换可以将一个复杂信号分解为多个简单的正弦波信号，这有助于理解信号的组成以及不同成分对信号整体的影响。

4. **信号重建**：通过逆傅里叶变换，可以从频域信息重建原始信号，表明频域与时域之间的深刻联系

2.1.2 傅里叶系数的物理含义

傅里叶系数是用来描述周期信号在频域中的分布特性。对于一个周期信号 $f(t)$，可以通过傅里叶级数将其表示为一系列正弦和余弦函数的线性组合：

$$f(t) = \sum_{n=-\infty}^{\infty} C_n e^{jn\omega_0 t}$$

其中 C_n 是傅里叶系数，ω_0 是基本频率。每个傅里叶系数 C_n 代表了频率为 $n\omega_0$ 的正弦波在原信号中的贡献。物理意义：

1. **频域表示**：傅里叶系数提供了信号在频域中的具体频率成分，反映了信号的周期性特征。高频成分和低频成分的强弱，可以通过傅里叶系数的大小来判断。

2. **能量分布**：在物理上，傅里叶系数的平方与信号在对应频率下的能量成正比，能量集中在特定频率的信号（如正弦波）其对应的傅里叶系数会比较大，而噪声信号则相对较小。

3. **调制与传输**：在通信系统中，傅里叶系数也反映了信号调制过程的特征，调制后的信号在不同频率上的傅里叶系数能够表征信息的传输和频谱利用情况。

2.1.3 傅里叶变换的应用

1. **频域分析：**
    - 图像可以看作是一个二维信号，傅里叶变换能够将其从空间域转换到频域。频域中的每个点表示图像在特定频率下的强度与相位信息。
    - 频域分析能够帮助我们理解图像的结构特征。例如，高频成分通常对应于图像中的边缘和细节，而低频成分则反映了图像的光照和颜色变化。
2. **图像滤波：**
    - 通过傅里叶变换，可以设计各种滤波器（如低通滤波器、高通滤波器和带通滤波器），用于图像的平滑、锐化或噪声去除。
    - 在频域中，对应的滤波操作可以通过乘以特定的频域滤波器来实现。例如，低通滤波器可以去除高频噪声，从而使图像更平滑。
3. **图像压缩：**
    - 傅里叶变换在图像压缩中也发挥着重要作用，例如 JPEG 压缩技术使用离散余弦变换（DCT），这是一种特殊形式的傅里叶变换。
    - 通过对图像进行频域变换，可以识别出重要的频率成分，并去除不重要的部分，从而达到压缩数据量的目的。

2.1.4 傅里叶系数的物理意义

1. **图像特征提取：**
    - 在频域中，傅里叶系数反映了图像各个频率成分的贡献。通过分析这些系数，可以提取图像的特征，例如纹理和边缘。
    - 对于不同的图像，傅里叶系数的分布会有所不同，这可以用于分类或识别任务，例如在计算机视觉中的物体识别。
2. **模式识别：**
    - 在模式识别和机器学习中，傅里叶系数可以作为特征向量来训练模型。通过频域特征，模型能够更好地捕捉到图像的结构信息，提高识别的准确性。

3. **图像重建：**

- 通过逆傅里叶变换，可以将频域信息转换回空间域，从而实现图像的重建。这一过程在图像恢复和去噪中非常重要。

2.2 实验内容二

2.2.1 顺时针旋转

Step1: 获取图像中心点

图像的中心点可以通过图像的宽度和高度来计算，即：

$$center = \left(\frac{w}{2}, \frac{h}{2}\right)$$

其中 w 是图像的宽度，h 是图像的高度。

Step2: 计算旋转矩阵

在实际应用中，我们通常需要围绕图像的某个点（如中心点）进行旋转，因此需要对旋转矩阵进行平移。旋转矩阵的一般形式为：

$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & tx \\ \sin(\theta) & \cos(\theta) & ty \end{bmatrix}$$

其中 tx, ty 是平移量。

Step3: 计算旋转后的图像尺寸在旋转图像时，原图像的尺寸会发生变化，可能会出现图像被裁剪的情况。为了避免这种情况，我们需要计算旋转后图像的宽度和高度。计算方法涉及到旋转矩阵中余弦和正弦值的绝对值：

$$\text{new\_w} = h \cdot |\sin(\theta)| + w \cdot |\cos(\theta)|$$

$$\text{new\_h} = h \cdot |\cos(\theta)| + w \cdot |\sin(\theta)|$$

这些公式帮助我们找到旋转后图像的最大边界。

Step4: 平移调整在旋转后，由于新图像的中心位置可能与原中心不一致，因此需要调整旋转矩阵的平移部分，使得旋转后的图像仍然以新的中心为基准。调整后的平移量计算如下：

$$M[0,2] += \left(\frac{\text{new\_w}}{2} - \text{center}[0]\right)$$

$$M[1,2]+=\left(\frac{\text{new\_h}}{2}-\text{center}[1]\right)$$

Step5: 图像变换最后，使用 cv2.warpAffine() 函数应用旋转矩阵进行仿射变换。该函数会根据旋转矩阵 M 和目标图像大小来重新计算每个像素的坐标，并将原图像中的像素映射到新图像中。

2.2.2 插值放大

最近邻插值放大：

最近邻插值是一种简单的图像插值方法，计算公式为：

$$I'(x,y)=I\left(\lfloor x\rfloor,\lfloor y\rfloor\right)$$

这里 $I(x,y)$ 是原图像的像素值，(x,y) 是放大后图像的坐标。

双线插值放大：

双线性插值是一种考虑周围四个像素的插值方法，其计算公式为：

$$I'(x,y)=(1-a)(1-b)I(0,0)+a(1-b)I(1,0)+(1-a)bI(0,1)+abI(1,1)$$

其中，a 和 b 是根据目标位置在原图像中相对位置计算得到的权重。

2.3 实验任务三

Step1: 进行傅里叶变换

使用 OpenCV 的 cv2.dft() 函数进行离散傅里叶变换。原理如下：

$$F(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)\cdot e^{-2\pi i\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

Step2: 移动频率原点

傅里叶变换后的结果通常将低频成分放在左上角，而高频成分放在右下角。为了更直观地查看频谱，使用 np.fft.fftshift() 将频率原点移至图像中心。这一步将低频成分移至频谱的中心，使得后续分析更方便。频率原点移动的原理可以通过以下公式表示：

$$F_{\text{shift}}(u,v)=F(u,v)e^{-j2\pi\left(\frac{N}{N_0}u+\frac{M}{M_0}v\right)}$$

其中 N 和 M 分别是图像的高度和宽度。

Step3: 计算幅度谱

幅度谱是频域图像的重要特征,表示各个频率成分的强度。cv2.magnitude() 函数计算复数的幅度，公式为：

$$|F(u,v)| = \sqrt{R(u,v)^2 + I(u,v)^2}$$

其中 $R(u,v)$ 和 $I(u,v)$ 是复数的实部和虚部。接着对幅度谱进行归一化和对数变换，以增强对比度。

3 实验结果

首先利用 python 里面的 OpenCV 库里的函数获得灰度图如下：

图 2: 灰度图

3.1 实验一

见实验内容。

3.2 实验二

3.2.1 顺时针旋转

将任务图顺时针旋转 30 度后得到图 3。

图 3: 旋转过后的灰度图

3.2.2 插值放大

最近邻插值和双线插值的结果如下：

图 4: 最近邻插值 x2

图 5: 最近邻插值 x4

图 6: 双线插值 x2

图 7: 双线插值 x4

将图片局部放大之后可以得出：

最近邻插值在放大图像时，会选择离目标像素最近的原始像素值进行填充。这种方法相对简单，计算速度较快，但放大后的图像可能出现明显的块状效应和锯齿状边缘。

双线性插值通过对目标像素周围的四个像素进行加权平均来计算新像素的值，能够提供更加平滑的结果。

3.3 实验三

图片的傅里叶变换图像如图 8

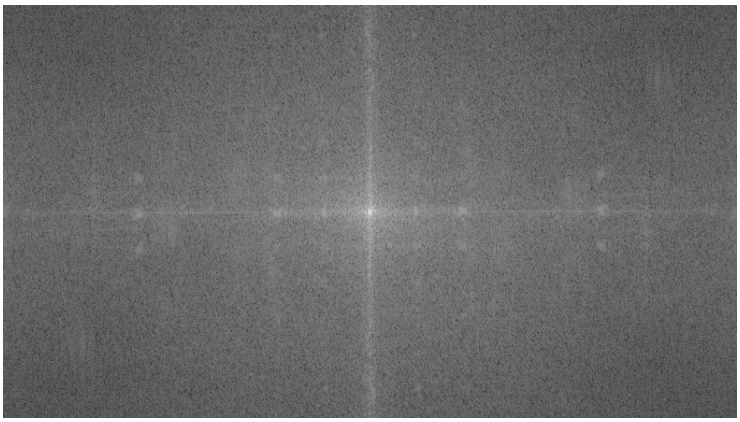

图 8: 傅里叶变换图像

由图可得，原图片中高频成分低频成分都有，且低频成分较多，图上亮度较大。

A 代码

Listing 1: 旋转图片 rotate.py

```python
import cv2
import numpy as np

# 读取彩色 BMP 图像
img = cv2.imread("lab.bmp")

# 将图像转换为灰度图
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 保存灰度图
cv2.imwrite("gray_image.bmp", gray_img)

# 获取图像中心
(h, w) = gray_img.shape[:2]
center = (w // 2, h // 2)

# 计算旋转角度
angle = -30
M = cv2.getRotationMatrix2D(center, angle, 1.0)

# 计算旋转后的图像的大小
cos = np.abs(M[0, 0])
sin = np.abs(M[0, 1])
new_w = int((h * sin) + (w * cos))
new_h = int((h * cos) + (w * sin))

# 调整旋转矩阵，使其以新尺寸为基准
M[0, 2] += (new_w / 2) - center[0]
M[1, 2] += (new_h / 2) - center[1]

```

```
# 旋转图像
rotated_img = cv2.warpAffine(gray_img, M, (new_w,
    new_h))

# 保存旋转后的图像
cv2.imwrite("gray_rotated_30.bmp", rotated_img)

# 显示旋转后的图像
cv2.imshow("Rotated Image", rotated_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Listing 2: 插值放大 enlarge.py

```
import cv2

# 读取灰度 BMP 图像
gray_img = cv2.imread("gray_image.bmp", cv2.
    IMREAD_GRAYSCALE)

# 放大倍数
scaling_factors = [2, 4]

# 分别使用最近邻和双线性插值放大图像
for factor in scaling_factors:
    # 最近邻插值
    resized_nn = cv2.resize(gray_img, (gray_img.shape
        [1] * factor, gray_img.shape[0] * factor),
        interpolation=cv2.INTER_NEAREST)
    cv2.imwrite(f"gray_image_resized_nn_{factor}x.bmp"
        , resized_nn)

    # 双线性插值
    resized_bilinear = cv2.resize(gray_img, (gray_img.
```

```
        shape[1] * factor, gray_img.shape[0] * factor),
         interpolation=cv2.INTER_LINEAR)
    cv2.imwrite(f"gray_image_resized_bilinear_{factor}
        x.bmp", resized_bilinear)

    # 显示放大后的图像
    cv2.imshow(f"Nearest Neighbor Resize {factor}x",
        resized_nn)
    cv2.imshow(f"Bilinear Resize {factor}x",
        resized_bilinear)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Listing 3: 傅里叶变换 ft.py

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

# 读取灰度图像
gray_img = cv2.imread("gray_image.bmp", cv2.
    IMREAD_GRAYSCALE)

# 进行傅里叶变换
dft = cv2.dft(np.float32(gray_img), flags=cv2.
    DFT_COMPLEX_OUTPUT)

# 将傅里叶变换结果移至频率原点
dft_shift = np.fft.fftshift(dft)

# 计算幅度谱并进行归一化
magnitude_spectrum = cv2.magnitude(dft_shift[:,:,0],
    dft_shift[:,:,1])
magnitude_spectrum = np.log(magnitude_spectrum + 1)  #
```

```
    防止对数为零
magnitude_spectrum = cv2.normalize(magnitude_spectrum,
    None, 0, 255, cv2.NORM_MINMAX)

# 保存幅度谱图像
cv2.imwrite("magnitude_spectrum.bmp",
    magnitude_spectrum.astype(np.uint8))

# 显示原图和幅度谱
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("原始灰度图像")
plt.imshow(gray_img, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("幅度谱")
plt.imshow(magnitude_spectrum, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```