**目 录**

[1. 设计目的 1](#_Toc7601)

[2. 设计原理 1](#_Toc29669)

[2.1. 二维傅立叶变换的基本原理 1](#_Toc2658)

[2.2. 傅立叶变换的旋转性质 2](#_Toc28174)

[2.3. 傅立叶变换的位移性质 2](#_Toc13906)

[2.4. 傅立叶变换的缩放性质 3](#_Toc2461)

[3. 设计过程 3](#_Toc14762)

[3.1. 界面设计 3](#_Toc24870)

[3.2. 控制属性设计 4](#_Toc25462)

[3.3. 功能设计 5](#_Toc21293)

[3.4. 设计中的关键技术 6](#_Toc15687)

[3.4.1. 技术问题描述 6](#_Toc15275)

[3.4.2. 技术问题解决方案 7](#_Toc4376)

[4. 设计结果 7](#_Toc19187)

[5. 自我总结 11](#_Toc11053)

[6. 附录 12](#_Toc27787)

# 设计目的

设计一款数字图像处理软件，利用MATLAB App Designer开发，演示傅立叶变换的基本性质，如旋转、位移和缩放等操作。该软件旨在实现以下功能：导入任意图像并进行傅立叶变换处理，展示旋转、位移和缩放等变换对图像频谱的影响；提供交互式界面，用户可以选择不同的傅立叶变换模式并实时查看变换后的图像频谱变化；设计简单易用的操作区，包括导入图片、选择变换模式、处理图像和重置等功能，确保用户能够便捷使用并快速掌握傅立叶变换的原理及其应用。

# 设计原理

## 二维傅立叶变换的基本原理

二维傅立叶变换将一个空间域图像转换为频域图像，从而表示图像的频率成分。对于一个二维图像 ，如图1，其二维傅立叶变换 表示为：

其中， 是原始图像，是傅立叶变换后的频谱， 和 分别是空间域和频域中的坐标，是虚数单位。频域中的每个点 表示图像在对应频率 下的强度信息，低频信息通常对应图像的平滑区域，而高频信息则代表图像的细节和边缘如图2。通过傅立叶变换，我们可以分析图像中的频率成分，进一步处理图像。



图 1 原图像

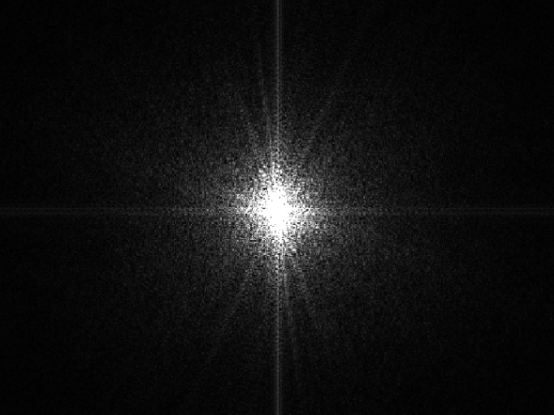


图 2 变换后图像

## 傅立叶变换的旋转性质

傅立叶变换具有旋转不变性，即图像在空间域中旋转一个角度后，其频域的旋转角度也与之相同。假设图像 旋转一个角度 ，其傅立叶变换 也将旋转相同的角度，如图3。具体公式为：

根据傅立叶变换的旋转性质，频谱的旋转角度与空间域图像的旋转角度一致，具体可表示为：

旋转后的频谱坐标和可以表示为：

这表示傅立叶变换后的频谱与空间域图像的旋转角度一致。因此，频谱的旋转和图像的旋转是相对应的，旋转角度保持一致。

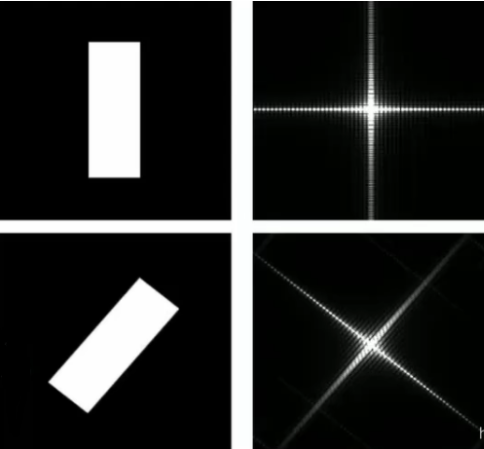


图 3 傅里叶变换的旋转性

## 傅立叶变换的位移性质

位移性质描述了图像平移对频域的影响。具体来说，如果图像 在空间域中平移了一个距离 ，其频谱 会发生相位的变化，但幅度保持不变。平移变换的数学表达式为：

这意味着图像的平移在频域中只会引起频谱的相位变化，频谱的幅度不变。

## 傅立叶变换的缩放性质

缩放性质指的是图像在空间域中的缩放操作如何影响其频域表示。当图像在空间域中进行缩放时，频谱会进行相应的缩放。假设图像被缩放了一个因子，即图像的尺寸被放大或缩小。此时，频谱的频率成分会被缩放为。具体地，缩放变换的公式为：

这一性质表明，在空间域中缩小图像会使频谱的频率成分扩展，而放大图像会使频谱的频率成分压缩。

傅立叶变换的旋转、位移和缩放性质为图像分析提供了强大的理论工具。通过在频域中对图像进行这些操作，能够直观地理解图像在不同变换下的变化规律。利用这些性质，本软件实现了对图像傅立叶变换的动态演示，帮助用户深入理解图像频域中的变换效果。

# 设计过程

## 界面设计

界面设计采用了MATLAB App Designer中的常用控件，旨在实现图像处理的可视化和交互性。界面设计的布局和控件选择是为了简化操作流程，使用户能够直观地观察到图像和傅立叶变换后的频谱效果。界面分为三个主要部分：图像显示区、操作区和控制区。

图像显示区：界面上分为两部分，左侧为“原始图像”区域，右侧为“处理后图像”区域。这里使用了Axes控件（app.axes1和app.axes2），分别展示原始图像和经过傅立叶变换处理后的图像或频谱。用户可以清晰地对比变换前后的效果。

操作区：包括文件导入、傅立叶变换模式选择、处理按钮和重置按钮。通过Button控件实现图像导入、处理和重置等功能。PopupMenu控件用于选择傅立叶变换的处理模式（旋转、位移、缩放）。通过这些操作，用户可以选择不同的处理模式，并实时查看频谱的变化。

设计思想：该布局旨在保持界面的简洁性和易用性，同时通过直观的控件布局，使得用户能够快速理解操作步骤并查看处理结果。

## 控制属性设计

每个控件的设计和属性设置都是为了增强用户体验，确保界面美观且功能实用。

图像显示区（Axes控件）：app.axes1和app.axes2分别用来显示原图和处理后的图像。在显示处理后的频谱时，我们设置了cla命令清除图像区域，以保证频谱的实时更新。Visible属性确保图像区域可见。

按钮控件（PushButton）：图像导入、处理图像和重置按钮分别设置了BackgroundColor、FontSize等属性，使得按钮清晰可见且易于操作。为了确保界面的美观，每个按钮都是对其的如图4。在按钮点击后，我们通过Enable属性控制按钮的可用状态，避免重复操作。



图 4 按钮控件

下拉菜单（PopupMenu）：app.popupmenu1用于选择傅立叶变换的模式。每个变换模式（旋转、位移、缩放）都是通过strcmp函数判断用户选择来确定的，具体的处理会根据选择动态展示。

## 功能设计

软件的功能设计分为多个模块，以便于管理和扩展。

图像导入模块：通过pushbutton1\_Callback函数，用户点击“导入图像”按钮后，使用uigetfile选择本地图像文件。该功能支持多种图像格式（如JPG、PNG、BMP）。图像被导入后，会在app.axes1中显示。代码如下。

function pushbutton1\_Callback(app, event)

[filename, pathname] = uigetfile({'\*.jpg;\*.png;\*.bmp', 'Image Files'}, '选择图像');

if isequal(filename, 0)

return;

end

img = imread(fullfile(pathname, filename));

imshow(img, 'Parent', app.axes1);

end

傅立叶变换处理模块：在“选择处理模式”下拉菜单如图5，用户可以选择“旋转”、“位移”或“缩放”。当用户选择某个模式并点击“处理图像”按钮后，程序会根据选择执行相应的傅立叶变换，并显示变换后的频谱图。旋转：利用imrotate函数旋转图像频谱。位移：利用imtranslate函数实现频谱的平移。缩放：利用imresize函数进行频谱缩放。处理结果会显示在app.axes2中，并更新频谱图。

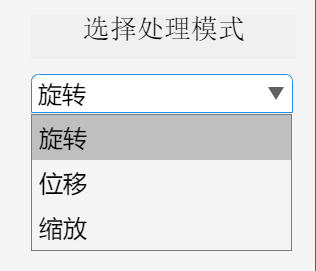


图 5 选择处理模式菜单

重置功能：点击“重置”按钮时，程序会清除处理后的频谱，恢复到原始状态，并使得用户可以重新选择新的图像进行处理。

处理图片功能：点击“处理图片”按钮时，程序会判断是否代入图片，如果没有导入图片则会发出提示如图6。

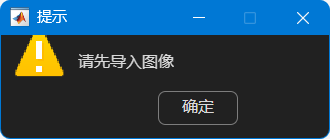


图 6 报警提示

## 设计中的关键技术

在进行多次图像处理时，如何有效管理图像的状态并确保每次操作都使用最新的处理图像，而非原始图像，是一个关键问题。为了解决这个问题，设计了一个机制，根据图像处理的状态来判断是使用原始图像进行处理，还是使用当前处理后的图像。下面是如何改进和优化代码来解决这个问题。

### 技术问题描述

在每次点击“处理图像”按钮时，我们需要根据当前的操作来判断是基于原始图像进行处理，还是使用上一次处理后的图像。特别是在旋转、位移和缩放等连续操作时，必须使用上一次的处理结果而不是每次都回到原始图像。以下代码是如何通过currentImg来管理图像状态和处理过程。

global img currentImg; % 添加currentImg用于连续操作

% 检查是否导入图像

if isempty(img)

msgbox('请先导入图像', '提示', 'warn');

return;

end

% 判断使用原始图像还是当前处理后的图像

if isempty(currentImg)

% 第一次处理，使用原始图像

processImg = im2gray(img);

else

% 连续操作，使用上次处理结果

if size(currentImg, 3) == 3

processImg = im2gray(currentImg); % 如果是彩色图像，转换为灰度图像

else

processImg = currentImg; % 如果是灰度图像，直接使用

end

end

### 技术问题解决方案

首次处理使用原始图像：在currentImg为空时，表示这是第一次进行图像处理，程序使用原始图像img来进行傅立叶变换处理。

连续操作使用处理后的图像：在用户进行连续操作时，currentImg会保存上次处理后的图像。当用户选择不同的操作（如旋转、位移、缩放）时，程序首先判断图像的格式（彩色或灰度），然后进行相应处理。特别是对于彩色图像，程序会将其转换为灰度图像后再进行傅立叶变换处理。

更新处理后的图像：每次图像处理完成后，我们将当前处理后的图像保存在currentImg中，并在下次操作时使用该图像。这确保了连续操作的正确性和效率。

图像处理功能：根据用户的选择（旋转、位移、缩放），程序会执行相应的傅立叶变换操作。每次操作后，变换结果通过fft2和fftshift计算频谱，并在axes2上显示结果。对于频谱的显示，使用log(abs(F) + 1)对频谱进行增强，使得细节更加清晰。

# 设计结果

点击“导入图片”按钮后，用户可以从文件选择框中选择本地的图像文件如图7。该按钮通过uigetfile打开文件浏览器，支持多种常见图像格式，如JPG、PNG和BMP等。选定文件后，图像会显示在左侧的原始图像区域如图8。

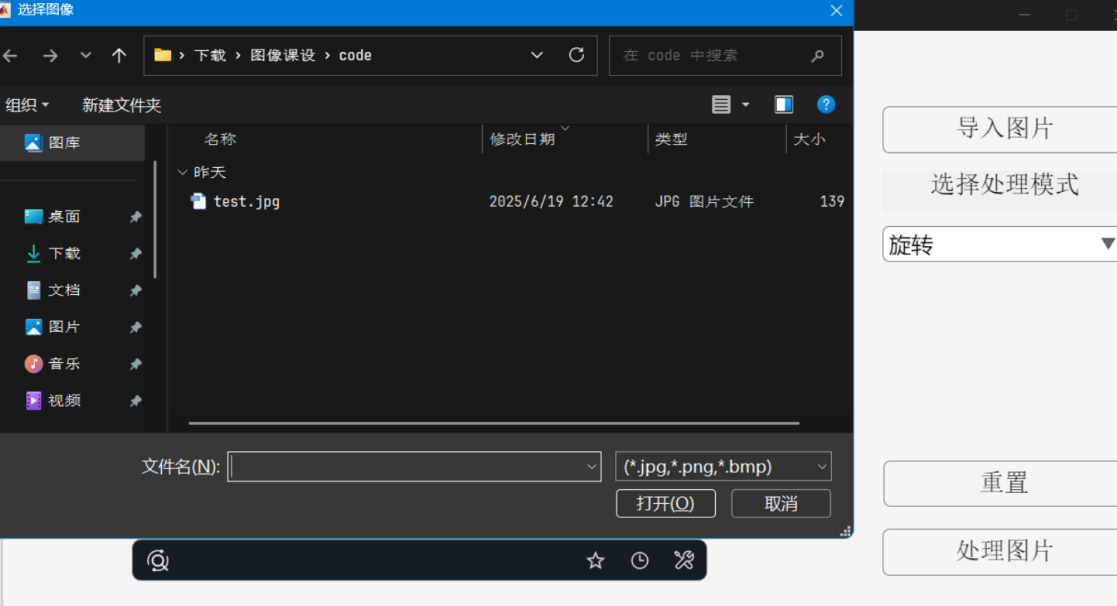


图 7 导入图片

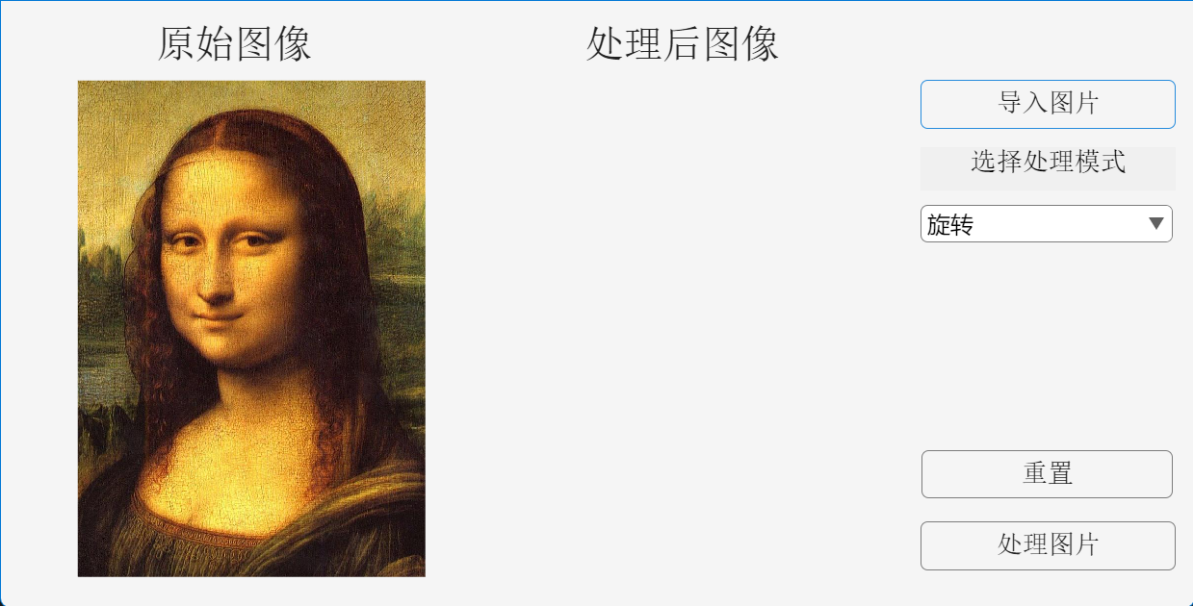


图 8 图片加载

在下拉菜单中，用户可以选择不同的傅立叶变换处理模式，如“旋转”、“位移”和“缩放”。用户选择不同的处理模式后，右侧的频谱图像会根据选择的操作进行更新。例如，选择“旋转”后，频谱图像会显示旋转后的频谱效果。

点击处理图像按钮后，程序根据下拉菜单选择的处理模式执行相应的图像处理操作，并在右侧显示处理后的频谱图像。每次点击“处理图像”按钮后，程序会执行所选的傅立叶变换，并更新处理后的图像或频谱如图9。

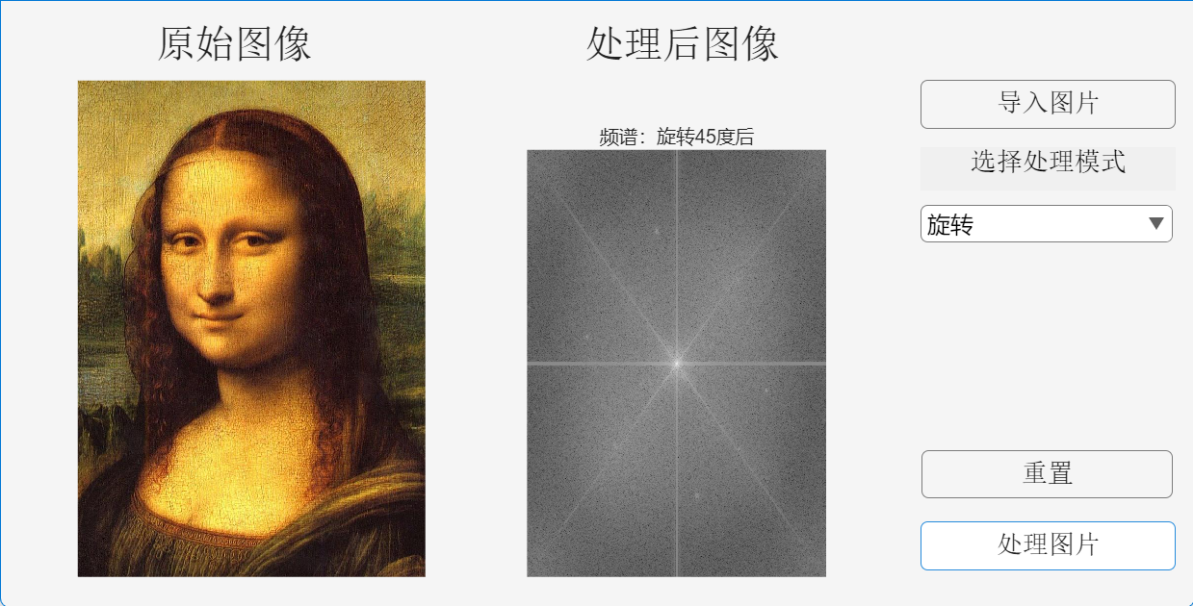


图 9 旋转后图像的傅里叶频谱

再次点击处理照片，即可实现傅里叶变换旋转90°效果如图10。

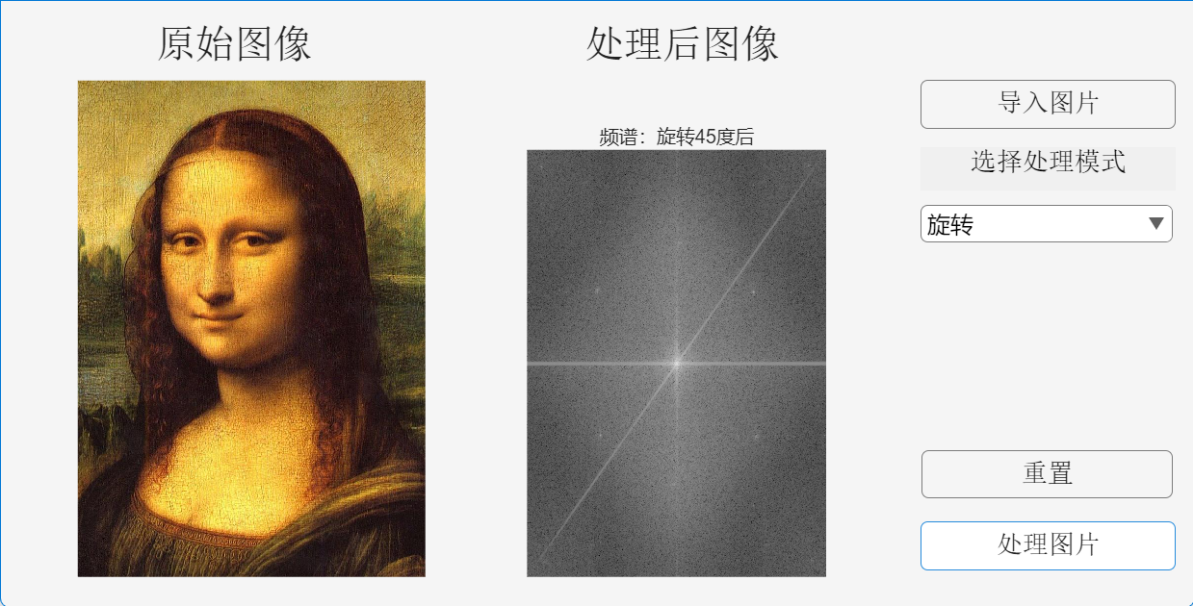


图 10 旋转后图像的傅里叶频谱

傅立叶变换处理（缩放）：用户选择“缩放”并点击“处理图像”，程序会缩小图像到80%并展示缩放后的频谱图像。为清晰验证缩放效果，多次点击处理图片，即可看到明显效果如图11。频谱中的细节得到了清晰的呈现。

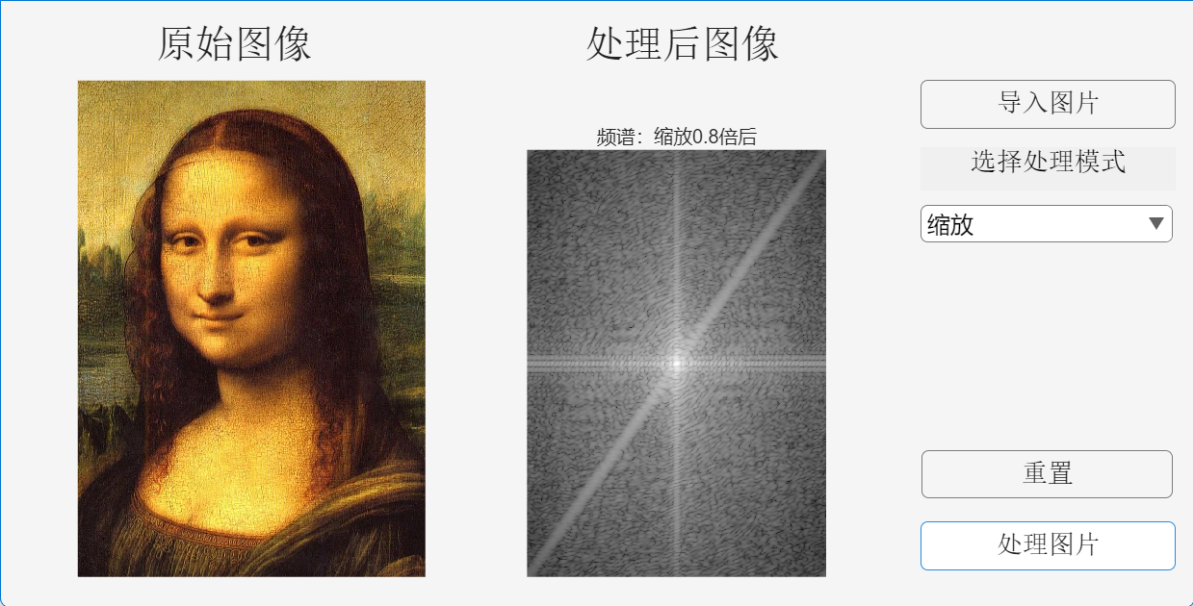


图 11 缩放后图像的傅里叶频谱

傅立叶变换处理（位移）：用户选择“位移”并点击“处理图像”，程序会平移图像50像素，为清晰验证位移效果，多次点击处理图片，并展示位移后的频谱图像如图12。频谱图像展示了平移操作对频率成分的影响。

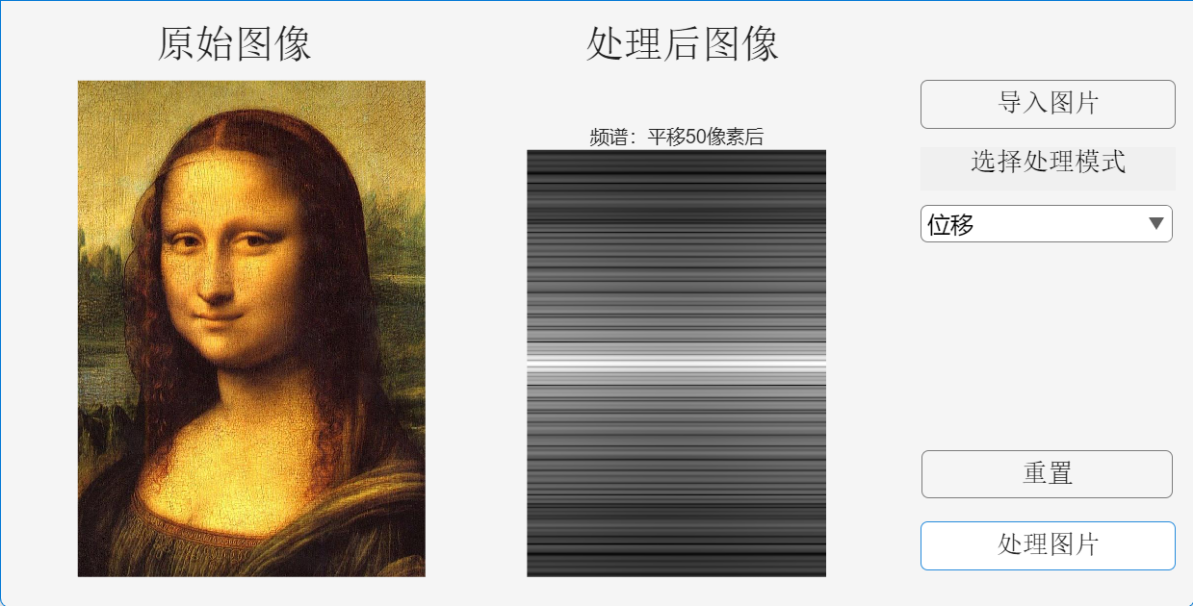


图 12 平移后图像的傅里叶频谱

重置按钮的功能是恢复到图像的初始状态。点击该按钮后，处理后图像区域都会恢复为初始状态，用户可以重新选择图像和变换操作，效果如图13。

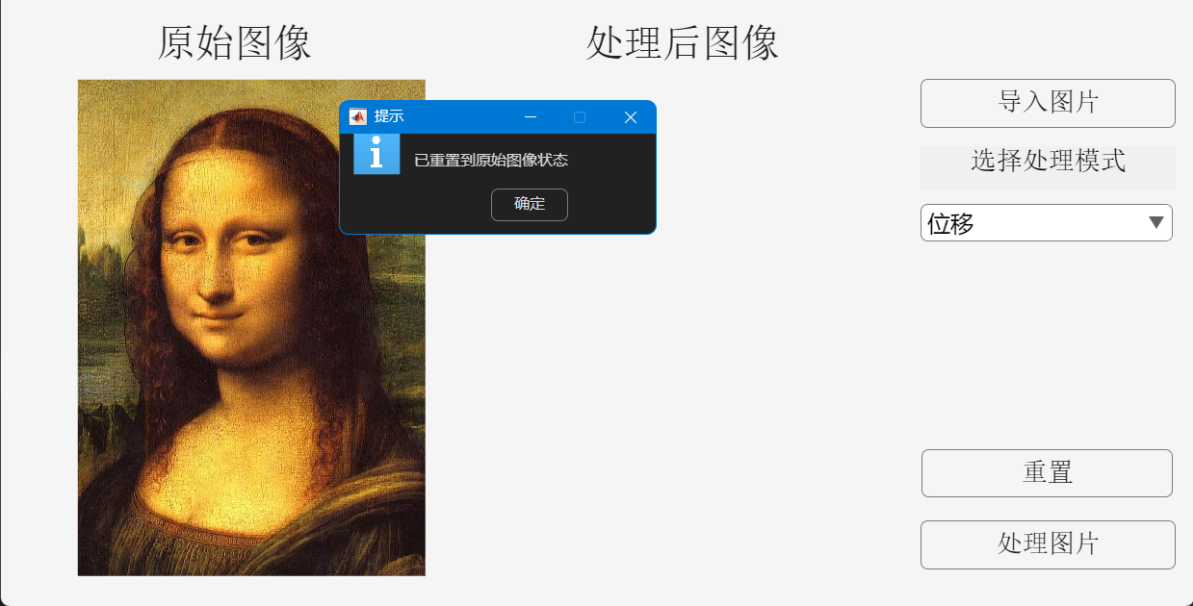


图 13 重置图片

# 自我总结

本次课程设计以傅立叶变换为核心，完成了图像处理软件的开发与实践。在理论学习阶段，我系统梳理了二维傅立叶变换的数学模型，重点分析了旋转、位移和缩放三大性质的数学表达与物理意义。通过公式推导与仿真验证，我深刻理解了频域与空域的对应关系：旋转操作中频谱与图像保持同步旋转的特性，位移操作中相位变化与幅度不变的规律，以及缩放操作中空间域与频域的反比关系。这些理论分析为后续的编程实现奠定了坚实基础。

在软件实现环节，我基于MATLAB App Designer平台构建了交互式处理系统。界面设计采用双视图布局，左侧显示原始图像，右侧实时展示频谱结果，通过按钮和下拉菜单实现功能控制。开发过程中遇到的核心问题是连续操作时的状态管理，例如多次旋转或缩放时需要基于前次结果迭代处理。通过引入全局变量动态存储图像状态，并增加灰度转换判断逻辑，最终实现了处理流程的连贯性。频谱可视化环节采用对数变换增强显示效果，解决了高频分量动态范围过大的问题。

通过本次实践，我的工程能力得到显著提升。一方面，掌握了GUI开发的完整流程，包括控件属性配置、回调函数编写和异常处理机制；另一方面，培养了调试复杂程序的思维习惯，例如通过分段验证确保傅立叶变换计算的准确性。此外，在查阅MATLAB官方文档和学术资料的过程中，我逐步养成了通过权威渠道解决问题的习惯。这些经验对后续开展更复杂的图像处理项目具有重要指导意义。

项目也存在若干待改进之处。当前系统仅支持单一图像串行处理，未来可扩展为多图像并行比对功能；交互方式上可增加滑动条控件，实现旋转角度、位移距离等参数的动态调节；算法层面可引入频域滤波等高级功能，丰富软件的应用场景。这些优化方向将为后续研究提供明确目标。总体而言，本次设计不仅巩固了理论知识，更通过完整的"理论-编程-调试-优化"闭环，显著提升了我的工程实践能力。

# 附录

classdef myFFT\_App < matlab.apps.AppBase

% Properties that correspond to app components

properties (Access = public)

figure1 matlab.ui.Figure

pushbutton3 matlab.ui.control.Button

pushbutton2 matlab.ui.control.Button

popupmenu1 matlab.ui.control.DropDown

text4 matlab.ui.control.Label

text3 matlab.ui.control.Label

text2 matlab.ui.control.Label

pushbutton1 matlab.ui.control.Button

axes2 matlab.ui.control.UIAxes

axes1 matlab.ui.control.UIAxes

end

methods (Access = private)

function edit\_angle\_CreateFcn(app, hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.

% hObject handle to edit\_angle (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

end

end

% Callbacks that handle component events

methods (Access = private)

% Code that executes after component creation

function untitled\_OpeningFcn(app, varargin)

% --- Executes just before untitled is made visible.

% Ensure that the app appears on screen when run

movegui(app.figure1, 'onscreen');

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app); %#ok<ASGLU>

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to untitled (see VARARGIN)

% Choose default command line output for untitled

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

global img;

img = [];

end

% Value changed function: popupmenu1

function popupmenu1\_Callback(app, event)

% --- Executes on selection change in popupmenu1.

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

contents = cellstr(get(hObject,'String'));

selected = contents{get(hObject,'Value')};

%cla(handles.axes2); % 清空 axes2

if strcmp(selected, '旋转') || strcmp(selected, '缩放') || strcmp(selected, '位移')

set(handles.pushbutton2, 'Visible', 'on');

else

set(handles.pushbutton2, 'Visible', 'off');

end

end

% Button pushed function: pushbutton1

function pushbutton1\_Callback(app, event)

% --- Executes on button press in pushbutton1.

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

% hObject handle to pushbutton1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

global img;

global img currentImg;

cla(handles.axes1); % 清空 axes1

cla(handles.axes2); % 清空 axes2

[filename, pathname] = uigetfile({'\*.jpg;\*.png;\*.bmp'}, '选择图像');

if isequal(filename,0)

return;

end

img = imread(fullfile(pathname, filename));

currentImg = []; % 重置连续操作状态

axes(handles.axes1);

imshow(img);

end

% Button pushed function: pushbutton2

function pushbutton2\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

global img currentImg; % 添加currentImg用于连续操作

% 检查是否导入图像

if isempty(img)

msgbox('请先导入图像', '提示', 'warn');

return;

end

% 清空显示区域

cla(handles.axes2);

title(handles.axes2, '');

contents = cellstr(get(handles.popupmenu1,'String'));

selected = contents{get(handles.popupmenu1,'Value')};

% 判断使用原始图像还是当前处理后的图像

if isempty(currentImg)

% 第一次处理，使用原始图像

processImg = im2gray(img);

else

% 连续操作，使用上次处理结果

if size(currentImg, 3) == 3

processImg = im2gray(currentImg);

else

processImg = currentImg;

end

end

% 执行图像处理

if strcmp(selected, '旋转')

% 旋转45°

rotated = imrotate(processImg, 45, 'bilinear', 'crop');

currentImg = rotated; % 保存当前状态

F = fftshift(fft2(rotated));

spectrum = log(abs(F) + 1);

axes(handles.axes2);

imshow(spectrum, []);

title('频谱：旋转45度后');

elseif strcmp(selected, '位移')

% 平移+50像素(x方向)

shifted = imtranslate(processImg, [50, 0]);

currentImg = shifted; % 保存当前状态

F = fftshift(fft2(shifted));

spectrum = log(abs(F) + 1);

axes(handles.axes2);

imshow(spectrum, []);

title('频谱：平移50像素后');

elseif strcmp(selected, '缩放')

% 缩放0.8倍

scale = 0.8;

scaled = imresize(processImg, scale);

% 居中填充

[H, W] = size(processImg);

canvas = zeros(H, W);

[h2, w2] = size(scaled);

row\_start = floor((H - h2)/2) + 1;

col\_start = floor((W - w2)/2) + 1;

canvas(row\_start:row\_start+h2-1, col\_start:col\_start+w2-1) = scaled;

currentImg = canvas; % 保存当前状态

F = fftshift(fft2(canvas));

spectrum = log(abs(F) + 1);

axes(handles.axes2);

imshow(spectrum, []);

title('频谱：缩放0.8倍后');

else

msgbox('请选择有效的图像处理方式', '提示', 'warn');

end

end

% Button pushed function: pushbutton3

function pushbutton3\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

global img currentImg;

% 检查是否已加载图像

if isempty(img)

msgbox('没有可重置的图像', '提示', 'warn');

return;

end

% 重置状态

currentImg = [];

% 重新显示原始图像

axes(handles.axes1);

imshow(img);

% 清空频谱显示

cla(handles.axes2);

title(handles.axes2, '');

% 可选：弹出提示信息

msgbox('已重置到原始图像状态', '提示', 'help');

end

end