

《计算机视觉（1）》实验报告

实验三 陷波滤波器

| 实验小组成员 (学号+班级+姓名) | 分工及主要完成任务 | 成绩 |
|---------------------------|-----------|----|
| 201800830004 18 数科 叶家辉 | 全部 | |
| | | |
| | | |

山东大学

2021 年 3 月

完成《数字图像处理》P185 页例 4.23 和 P186 页例 4.24 的编程实验，编程语言可以选择 Matlab, C, C++, OpenCV, Python 等。设计方案可参照教科书中的分析，也可以自行设计新的方案。

1、（P185 例 4.23）使用陷波滤波减少莫尔(波纹)模式。图 1 是来自扫描报纸的图像，它显示了突出的莫尔模式，设计一个布特沃斯陷波带阻滤波器消除图像中的莫尔条纹。



图1 莫尔模式的取样过的报纸图像

2、（P186 例 4.24）使用陷波滤波增强恶化的卡西尼土星图像。图 2 显示了部分环绕土星的土星环的图像。太空船第一次进入该行星的轨道时由“卡西尼”首次拍摄了这幅图像。垂直的正弦模式是在对图像数字化之前由叠加到摄影机视频信号上的 AC 信号造成的。这是一个想不到的问题，它污染了来自某些任务的图像。设计一种陷波带阻滤波器，消除干扰信号。

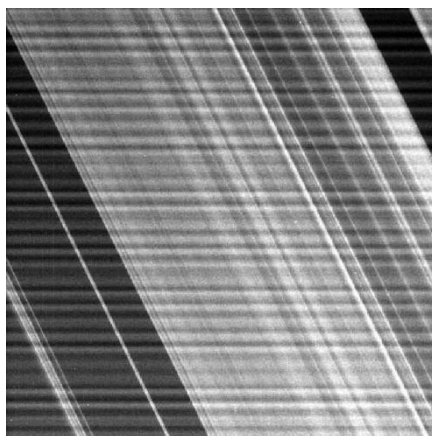


图2 近似周期性干扰的土星环图像，图像大小为 674×674 像素

原始图像的电子版图像在 Images 文件夹中。实验报告写在如下空白处，页数不限。

一、实验目的与原理

本次实验主要考察的是选择性滤波器中陷波滤波器的使用。陷波滤波器拒绝或通过人为定义的关于频率矩阵中心的一个邻域内的频率，可有效消除或增强频域上的特定部分，达到精准滤波的效果。在此之前，有必要先复习一下频域滤波的一般步骤。频域滤波一般包括以下步骤：

1. 对给定大小为 $M \times N$ 的图像进行零填充至大小为 $P \times Q$ ，记为 $f_p(x, y)$ 。一般地，取 $P = 2M$ ， $Q = 2N$ 。
2. 用 $(-1)^{x+y}$ 乘以 $f_p(x, y)$ ，以便将图像移到做变换后的中心，然后计算其 DFT，得到 $F(u, v)$ 。
3. 根据需要生成某个大小为 $P \times Q$ 的实对称滤波函数 $H(u, v)$ ，中心在 $(\frac{P}{2}, \frac{Q}{2})$ 处。计算阵列乘积 $G(u, v) = H(u, v) \times F(u, v)$ 。
4. 将处理后的频域图像进行还原： $g_p(x, y) = \{real[F^{-1}[G(u, v)]]\}(-1)^{x+y}$ 。
5. 从 $g_p(x, y)$ 的左上角截取 $M \times N$ 区域，得到最终处理结果 $g(x, y)$ 。

所以我们发现，频域滤波在操作层面所讨论的核心其实在于滤波函数 $H(u, v)$ 的构造。只要确定了 $H(u, v)$ 的形式，其它的只要按部就班就可以了。

零相移滤波器必须是关于原点对称的，所以一个中心位于 (u_0, v_0) 的陷波在位置 $(-u_0, -v_0)$ 必须有一个对应的陷波。陷波带阻滤波器一般可以写作一系列高通滤波器的乘积，它们的中心已被平移到陷波滤波器中心：

$$H_{NR}(u, v) = \prod_{k=1}^Q H_k(u, v) H_{-k}(u, v)$$

对于每一个滤波器，距离的计算方式如下：

$$D_k(u, v) = [(u - \frac{M}{2} - u_k)^2 + (v - \frac{M}{2} - v_k)^2]^{1/2} \quad (1)$$

$$D_{-k}(u, v) = [(u - \frac{M}{2} + u_k)^2 + (v - \frac{M}{2} + v_k)^2]^{1/2} \quad (2)$$

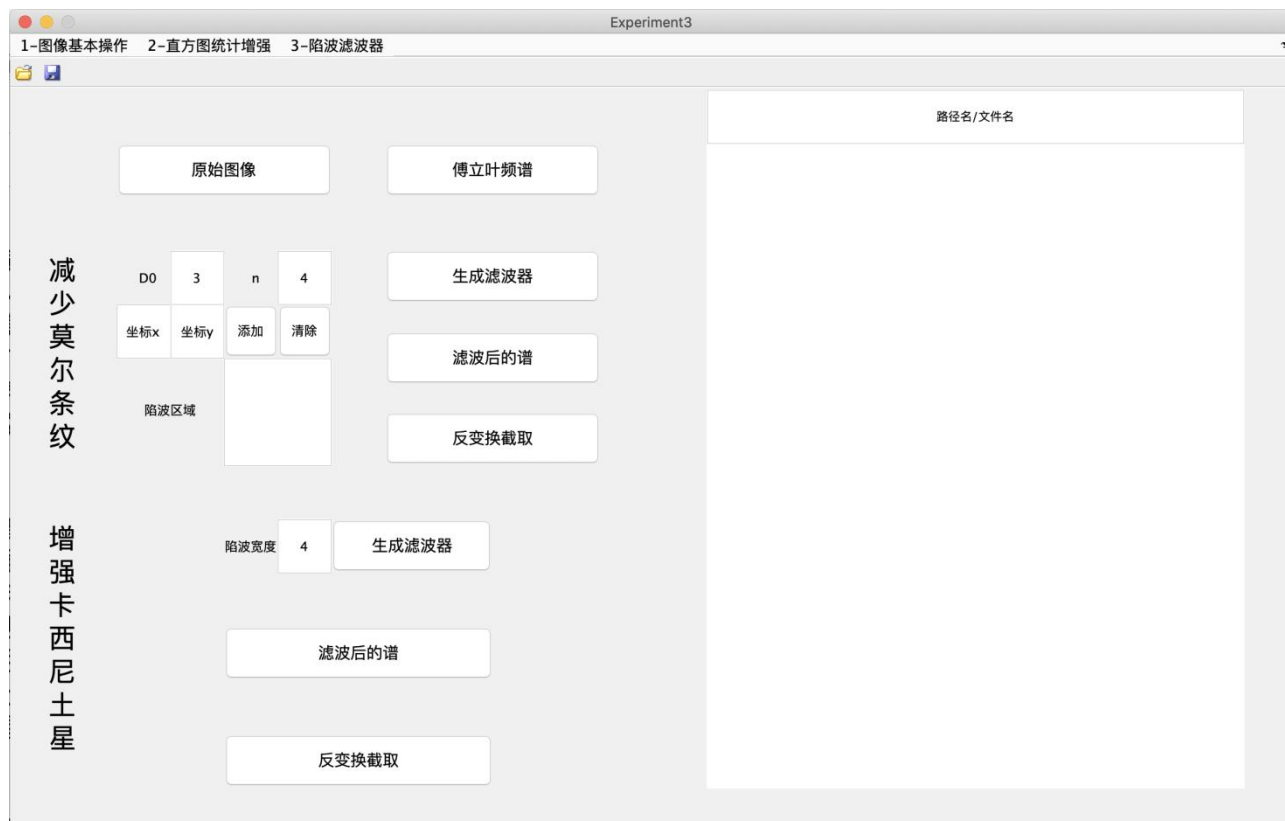
所以对于 n 阶巴特沃斯陷波带阻滤波器，可以得出：

$$H_{NR}(u, v) = \prod_{k=1}^Q \left[\frac{1}{1 + [D_{0k}/D_k(u, v)]^{2n}} \right] \left[\frac{1}{1 + [D_{0k}/D_{-k}(u, v)]^{2n}} \right] \quad (3)$$

例子中给出了参考值 $D_0 = 3$ ， $n = 4$ 。本实验中可以通过改变参数的值观察滤波效果的变化。

二、实验内容与步骤

2.1 界面总体设计与图像导入



本实验的 GUI 界面延续之前一贯的简洁明快的设计风格。同时为了集成计算机视觉 (1) 的所有实验项目，在 GUI 的顶部特别设计了导航栏，可以在同一窗口定位到不同的实验项目。点击“3-陷波滤波器”后即可得到本实验的整体操作界面。

和前几个实验一样，左上角有“打开文件”和“保存图像”两个图标。点击“打开文件”图标选择相应图片，即可在坐标图区 axes1 导入这张图片。“打开文件”按钮的回调函数如下图所示。前四行获取了所选定图片的文件名和路径名，再根据得到的路径名和文件名读取文件。后两行则把读取到的文件显示到坐标图区，并设定顶部的文本框显示图片路径名/文件名。

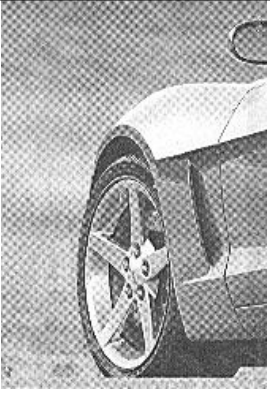
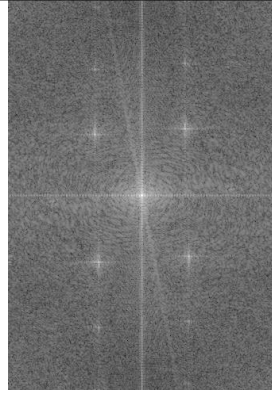
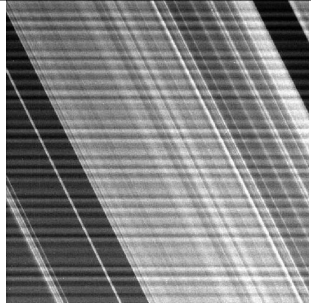
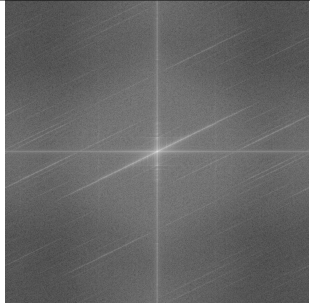
```
% -----  
function uipushtool1_ClickedCallback(hObject, eventdata, handles)  
% hObject    handle to uipushtool1 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)  
[filename,pathname]=uigetfile({'*.bmp;*.jpg;*.png;*.jpeg;*.tif'}, '请选择图片');  
str=[pathname filename];  
OriginalPic = imread(str);  
axes(handles.axes1);  
  
imshow(OriginalPic);  
set(handles.edit1, 'String', str);
```

2.2 原图像显示与傅立叶频谱绘制

为了方便将滤波后的图像与原图像进行对比，用户只要点击“原始图像”就可以显示图像原始的样子。回调函数如下图，非常简单，不再赘述。

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
file = get(handles.edit1,'string');
img = imread(file);
imshow(img);
```

傅立叶频谱可以直观展示一幅图像的特征，后面的操作也大多是基于此展开，所以让用户可以方便地得到当前图像的傅立叶频谱显得尤为重要。下面展示了本实验所使用的图片的傅立叶频谱。

| 原图像 | 傅立叶频谱 | 原图像 | 傅立叶频谱 |
|--|--|---|--|
|  |  |  |  |

绘制傅立叶频谱的回调函数如下图所示。首先读入图像，然后获取图像的宽度 M 和高度 N ，再进行傅立叶变换并进行零填充，使得图像大小变为 $2M \times 2N$ 。接下来我们需要将变换的原点移动到频率矩形的中心。一般我们可以在进行变换前将输入图像乘以 $(-1)^{x+y}$ ，不过通过 `fftshift()` 函数可以实现一样的效果。需要注意的是，移动居中变换与傅立叶变换不可互换，即 $F[(-1)^{x+y}f(x,y)]$ 等于 `fftshift(fft2(f))` 但不等于 `fft2(fftshift(f))`。虽然该移动像我们期望的那样完成了，可是在显示的时候只有亮点比较明显，其余区域大多是黑色的，这是因为该频谱中值的动态范围（ $0 \sim 204000$ ）与 8bit 显示相比要大得多，此时中心的明亮值占据主导地位。因此，我们可以进行对数变换后再显示。也就是

$$S = \log(1 + abs(Fc))$$

同时该图像中的值大多超出了 8bit 表示范围，这在图像显示和保存时会遇到麻烦，因此我们使用 `mat2gray()` 函数把灰度值限制在 $0 \sim 255$ 之间。

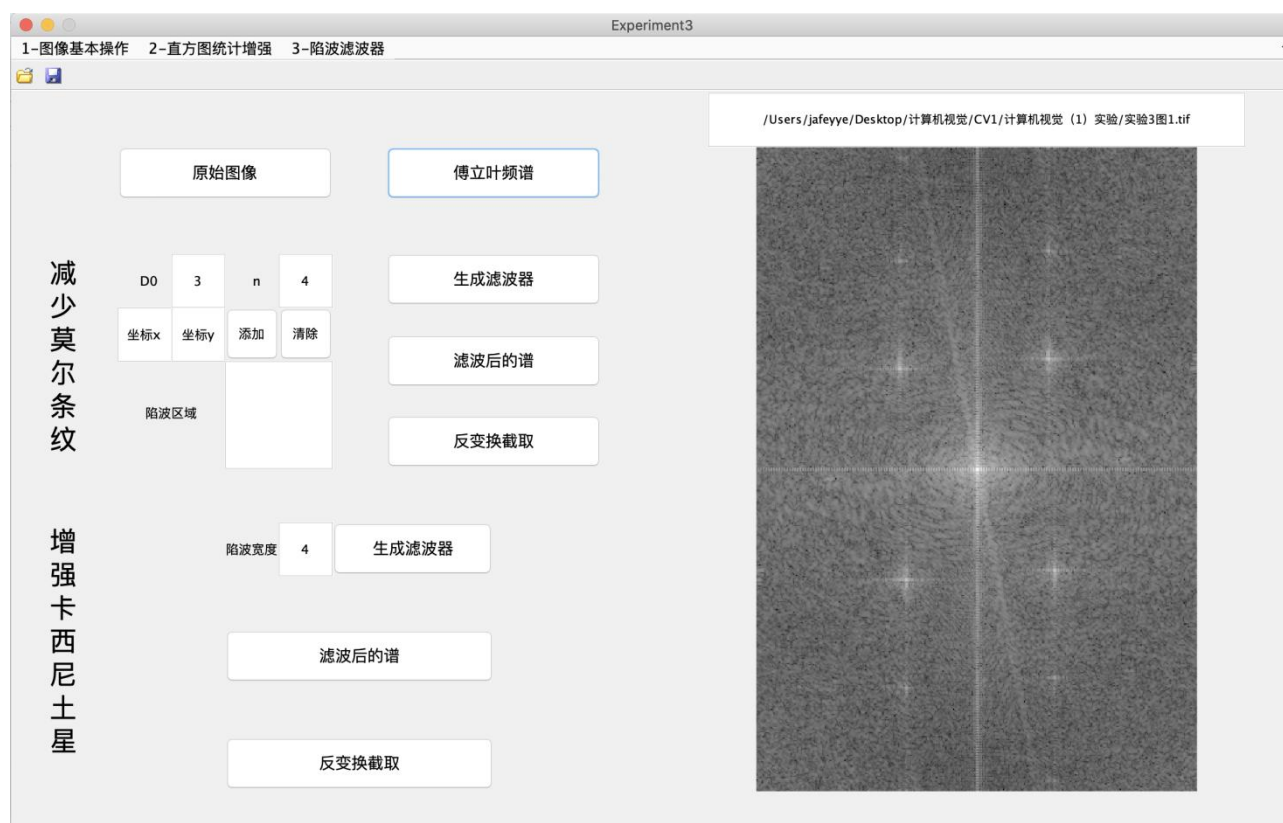

```

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton11 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
file = get(handles.edit1, 'string');
f = imread(file);
[M,N] = size(f);
F = fft2(f,2*M,2*N);
Fc = fftshift(F);
S = log(1+abs(Fc));
S = mat2gray(S);
imshow(S,[]);

```

2.3 减少莫尔条纹的实现

首先导入原图，这是一张扫描报纸的图像。可以发现，它显示出了突出的莫尔模式。接下来点击“傅立叶频谱”，就可以得到这张图像的谱。我们已经知道，作为周期函数的一个纯正弦的傅立叶变换是一对共轭对称的冲激。在这张谱中，这些对称的“类冲激”是莫尔模式的近似周期性的结果。我们可以利用陷波滤波器减少这些脉冲。



为了完成对巴特沃斯陷波带阻滤波器的设计，我们需要得到一系列参数，包括陷波半径 D_0 ，滤波器阶数 n 和陷波中心位置。其中，半径的选择应该能够包围所有脉冲能量，阶数的选择为给出陷波的适度过渡形状，而陷波中心位置则需要在谱的图上通过鼠标响应交互地进行选择。

本实验中给出了参考值 $D_0 = 3$, $n = 4$, 用户也可以自行修改文本框以尝试不同的参数。用户用鼠标点击谱中的点即可获得该点的坐标, 点击“添加”即可将该点添加到“滤波区域”文本框内, 点击“清除”也可以清除“滤波区域”文本框内的点。

下面是“添加”按钮的回调函数。为方便后续操作, 我们将文本框中的内容存储为矩阵形式。首先读入文本框原有的矩阵, 然后将当前坐标 x 和坐标 y 添加到最后一行。需要注意的是, 文本框 `edit` 默认只能显示一行, 如果要多行显示需要设置其最大显示行数。图中所示为设置了最大显示行数 100, 实际操作中一般不会超过 100 个陷波中心位置, 因此这样设置足以满足实操需求。

```
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton6 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
edit = str2num(get(handles.edit3, 'string'));
newx = get(handles.edit4, 'string');
newy = get(handles.edit7, 'string');
set(handles.edit3, 'max', 100);
edit = [edit; str2num(newx), str2num(newy)];
set(handles.edit3, 'string', num2str(edit));
```

接下来我们就可以根据所给定的参数完成滤波器的设计。“生成滤波器”的回调函数如下图所示。首先是准备工作, 包括读入图像并获取图像尺寸、获取参数 (包括陷波半径 D_0 , 滤波器阶数 n 和陷波中心位置)、将陷波中心位置坐标转化为中心化后的坐标。然后我们根据式(1)、(2)、(3), 通过两个 `for` 循环完成对整个 $P \times Q$ 内每个像素位置的操作, 最内层的 `for` 循环是为了将多个高通滤波器对进行累乘, 这样我们就根据前面的讨论生成了巴特沃斯陷波带阻滤波器。

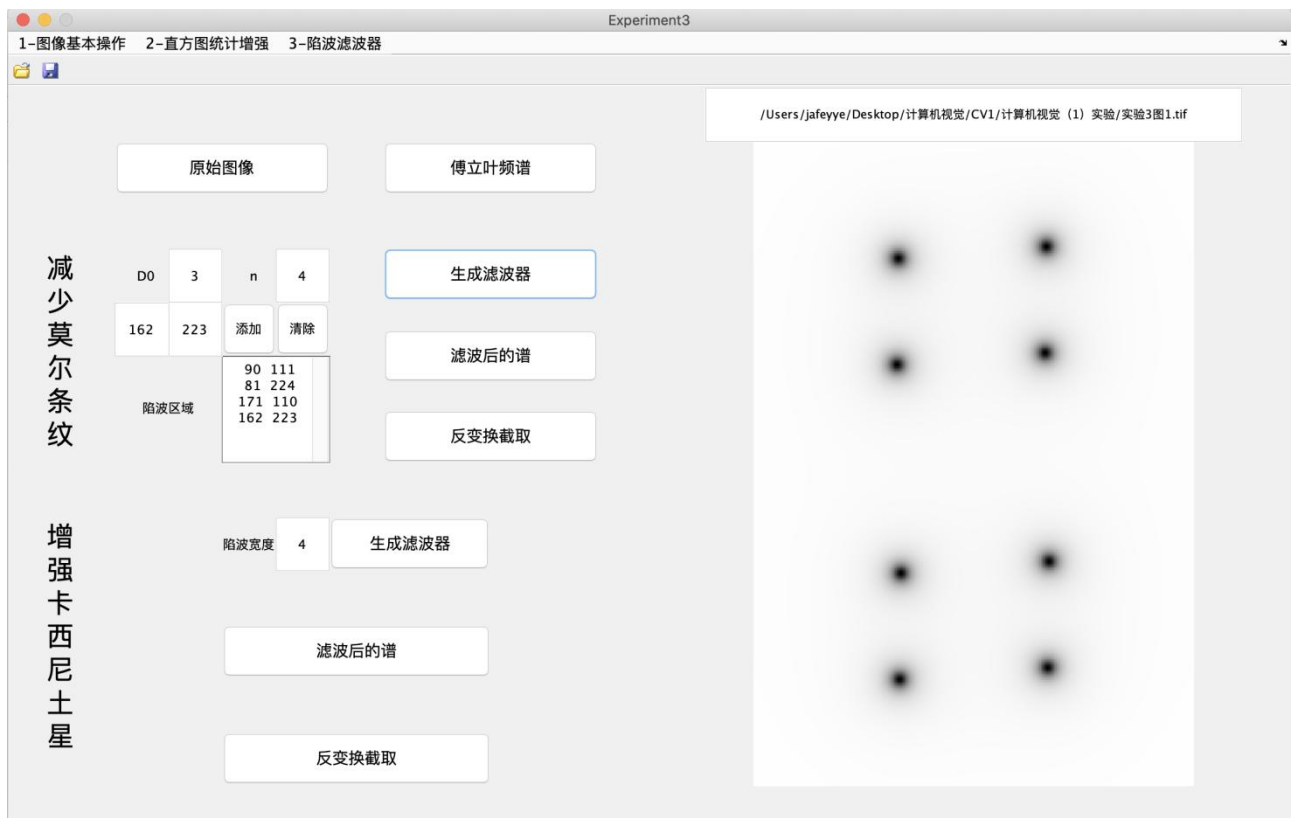
```

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton8 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
file = get(handles.edit1, 'string');
img = imread(file);
[M,N] = size(img);
D0 = str2num(get(handles.edit8, 'string'));
n = str2num(get(handles.edit9, 'string'));
uv = str2num(get(handles.edit3, 'string'));
uv(:,1) = uv(:,1)-M;
uv(:,2) = uv(:,2)-N;

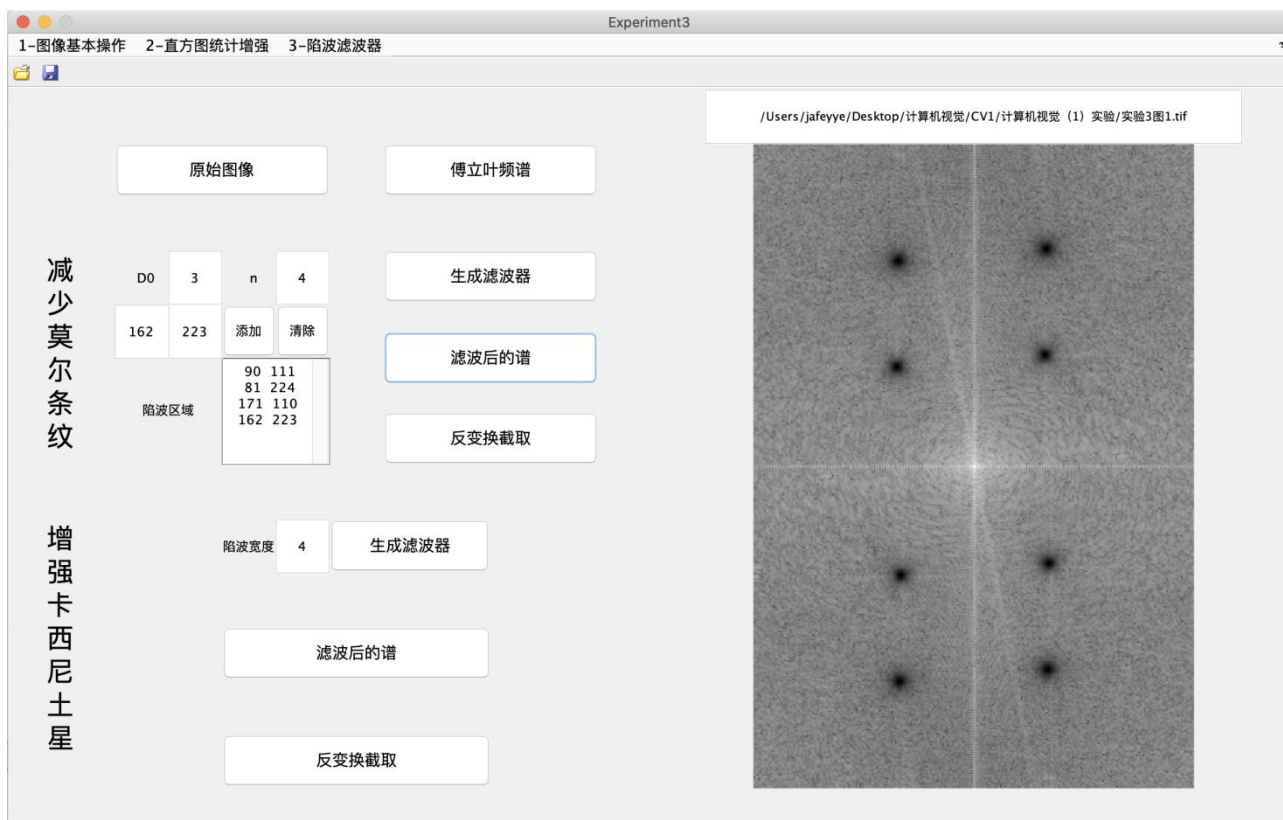
for u = 1:2*M
    for v = 1:2*N
        H(u,v) = 1;
        for i = 1:length(uv)
            Dk(u,v) = [(u-M-uv(i,1))^2+(v-N-uv(i,2))^2]^(1/2);
            D_k(u,v) = [(u-M+uv(i,1))^2+(v-N+uv(i,2))^2]^(1/2);
            H(u,v) = H(u,v)*(1/(1+(D0/Dk(u,v))^2*n))*(1/(1+(D0/D_k(u,v))^2*n));
        end
    end
end
imshow(H);

```

下图展示了生成的滤波器的图像。可以看出，之前设置的陷波中心附近的频率通过率较低，其他位置则较高，因此可以实现对干扰脉冲的去除。



接下来我们生成滤波后图像的傅立叶频谱。“滤波后的谱”的回调函数前面部分就是之前讨论过的生成傅立叶频谱、生成滤波器，只是最后多了一步将两个矩阵对应元素相乘。



最后我们需要对滤波完的频域图像进行反变换与还原，再截取左上象限区域得到最终图像。下图展示的“反变换截取”的回调函数包括了整个实验过程的代码。

前半部分就是之前讨论过的生成傅立叶频域矩阵和生成巴特沃斯陷波带阻滤波器。然后将频域矩阵与滤波器相乘。需要注意的是，前面我们为了加强显示效果，将变换后的频域信号转化为频谱并做了对数变换，之后为了展示滤波对频谱的改变作用又将加强后的频谱与滤波器相乘，但实际上，在滤波时我们只需要直接将中心化后的 F_c 与滤波器相乘即可。记得到的乘积矩阵为 R 。再对 R 做傅立叶反变换并取实部，记为 iR 。接下来我们需要将 iR 的每一个元素乘以 $(-1)^{x+y}$ ，为此我们生成一个因子矩阵 $h(x,y)$ ，使其 (x,y) 上的元素就等于 $(-1)^{x+y}$ ，然后将这个矩阵与 iR 相乘。需要注意的是，做“ $.*$ ”运算需要先把 iR 转化成 `double` 类型，保证前后数据类型一致。记结果为 `fillimage`。最后我们对其进行截取，取左上方宽度为 M ，高度为 N 的区域，即为处理后的最终图像。

和前面遇到的问题一样，该图像中的值大多超出了 8bit 表示范围，这在图像显示和保存时会遇到麻烦。我们可以直接使用“`imshow(final,[]);`”，这样图像在显示的时候就会比较正常，但是并不改变图片本身，因此保存的时候还是会出现问题。这里我们使用 `mat2gray()` 函数把图像的灰度值限制在 0 ~ 255 之间，这样也比较有利于后面的图像保存。


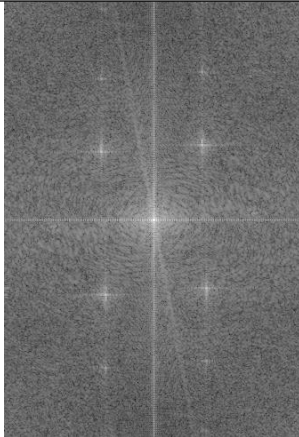
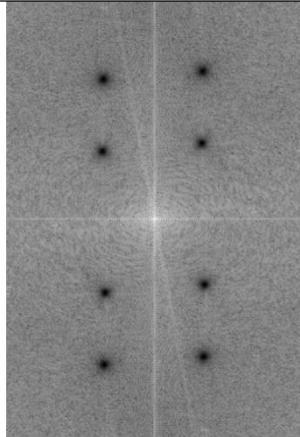

```

% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = get(handles.edit1,'string');
f = imread(file);
[M,N] = size(f);
F = fft2(f,2*M,2*N);
Fc = fftshift(F);
% S = log(1+abs(Fc));

[M,N] = size(f);
D0 = str2num(get(handles.edit8,'string'));
n = str2num(get(handles.edit9,'string'));
uv = str2num(get(handles.edit3,'string'));
uv(:,1) = uv(:,1)-M;
uv(:,2) = uv(:,2)-N;

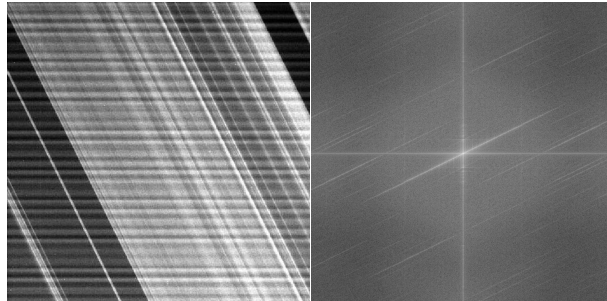
for u = 1:2*M
    for v = 1:2*N
        H(u,v) = 1;
        for i = 1:length(uv)
            Dk(u,v) = [(u-M-uv(i,1))^2+(v-N-uv(i,2))^2]^(1/2);
            D_k(u,v) = [(u-M+uv(i,1))^2+(v-N+uv(i,2))^2]^(1/2);
            H(u,v) = H(u,v)*(1/(1+(D0/Dk(u,v))^2*n))*(1/(1+(D0/D_k(u,v))^2*n));
        end
    end
end
R = Fc.*H;
iR = real(ifft2(R));
for x = 1:2*M
    for y = 1:2*N
        h(x,y) = (-1)^(x+y);
    end
end
fillimage = h.*double(iR);
final = fillimage(1:M,1:N);
final = mat2gray(final);
imshow(final,[]);

```

| 原图像 | 傅立叶频谱 | 滤波后的谱 | 反变换截取 |
|---|---|--|---|
|  |  |  |  |

2.4 增强卡西尼土星的实现

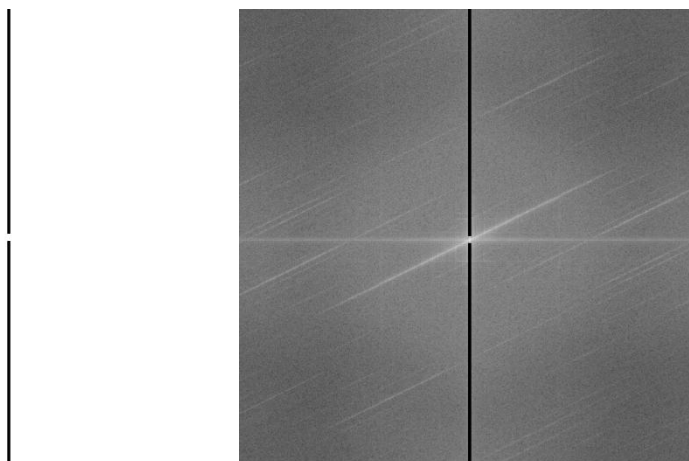
首先导入图像，这是一张近似周期性干扰的土星环图像。垂直的正弦模式是在对图像数字化之前由叠加到摄影机视频信号上的 AC 信号造成的。这些干扰很容易使用后处理方法进行校正。



首先需要生成滤波函数。通过观察傅立叶频谱我们发现垂直轴有一系列小能量脉冲，这些脉冲对应于近似正弦曲线的干扰。我们选择窄陷波矩形滤波器，它从最低频脉冲开始，扩展到垂直轴的剩余部分。下图展示了“生成滤波器”的回调函数，其中陷波的宽度是可以交互设置的，默认值为 4。

```
% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton12 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
file = get(handles.edit1, 'string');
f = imread(file);
[M,N] = size(f);
d = str2num(get(handles.edit11, 'string'));
flt = ones(size(2*M, 2*N));
flt(:, M-d:M+d) = 0;
flt(N-10:N+10, :) = 1;
imshow(flt);
```

下图展示了通过这种方法生成的滤波器的图像和滤波后谱的图像。

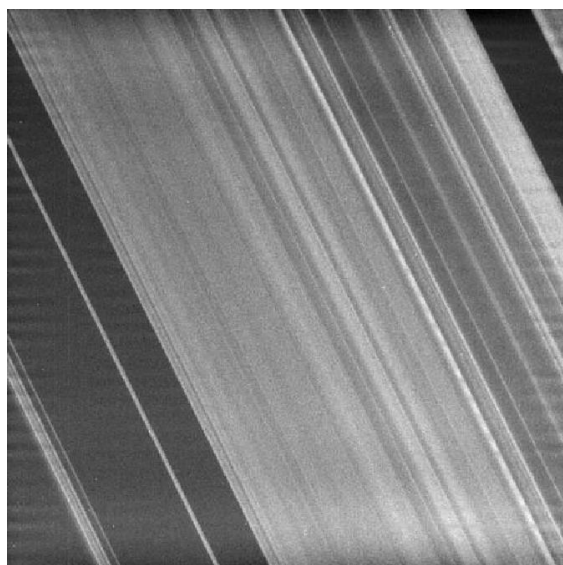


和莫尔条纹处理实验一样，只需要将生成的滤波器和频域矩阵相乘，再经过反变换还原到空间域图像，再进行裁剪就可以得到处理后的最终结果。回调函数如下图，具体过程不再赘述。



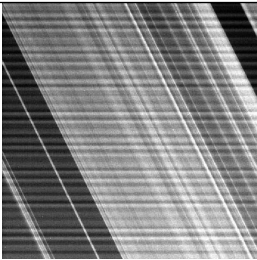
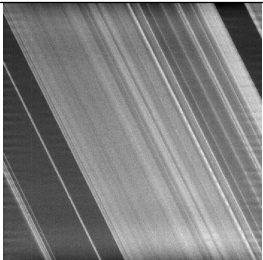









```
% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton14 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
file = get(handles.edit1, 'string');
f = imread(file);
[M,N] = size(f);
F = fft2(f,2*M,2*N);
Fc = fftshift(F);
% S = log(1+abs(Fc));
d = str2num(get(handles.edit11, 'string'));
flt = ones(2*M,2*N);
flt(:,M-d:M+d)=0;
flt(N-10:N+10,:)=1;
R = Fc.*flt;

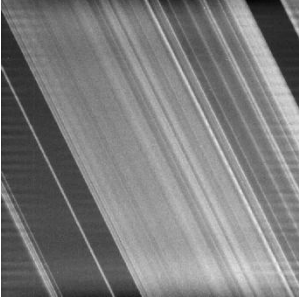
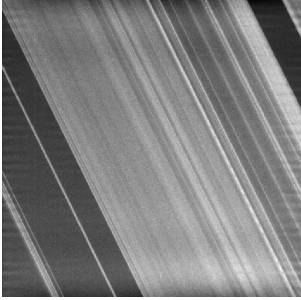
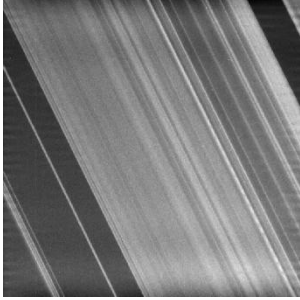
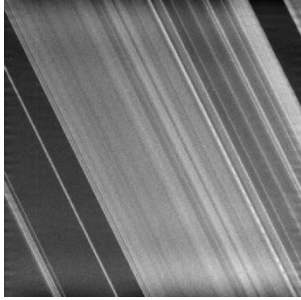
iR = real(ifft2(R));
for x = 1:2*M
    for y = 1:2*N
        h(x,y) = (-1)^(x+y);
    end
end
fillimage = h.*double(iR);
final = fillimage(1:M,1:N);
final = mat2gray(final);
imshow(final, []);
```

滤波的结果如下图所示。可以看出，效果还是比较不错的。



三、实验结果与总结

| 原图 | 滤波后 | 原图 | 滤波后 |
|---|---|--|---|
|  |  |  |  |
| | $D_0 = 1$ | $D_0 = 3$ | $D_0 = 5$ |
| $n = 2$ |  |  |  |
| $n = 4$ |  |  |  |
| $n = 6$ |  |  |  |

| $d = 2$ | $d = 4$ | $d = 6$ | $d = 8$ |
|---|---|--|---|
|  |  |  |  |