数字图像处理
第二次小作业报告

自96 曲世远 2019011455

Problem1

1.算法要点与理论原理

我认为本次作业第一题主要有以下几个要点：

1. 首先需要生成满足题目要求的二位波形图形，并且可以依据参数进行调节；
2. 针对题目要求的五种波形需要设计出对应于波形的参数调整UI界面；
3. 实现DFT运算函数并且保证运算速度；
4. 实现较为美观且准确的结果显示

针对以上我认为以及实现过程中遇到的算法要点，我才用了以下的算法原理加以解决：

1. 将波形产生函数进行整合，并基于$Wavetype$变量进行波形发生；
2. 由于如果采用但界面+隐藏UI设计会导致很大的重复代码量且不利于调整界面设计以及不够直观；因此我使用了$Matlab\ app\ designer$中的选项卡组容器，容纳不同波形选项时的参数调整组件。
3. 实现了基于矩阵的DFT运算函数以提高运算速度并且保证正确性，同时我引入了系统内置的FFT函数可以进行两种变换的比较，以验证正确性。
4. 由于surf并不能直接显示在$Matlab\ app\ designer$上，因此需要通过坐标图，并将surf加载到坐标图上。但由于$Matlab\ app\ designer$中的坐标图组件参数过少，并不能指定坐标轴的刻度显示精度，因此我设计了精确显示模式（可能会因为小数点后位数过多导致显示不美观）与非精确显示模式。

2.算法具体实现

```
%% generate origin figure
function Wave_generate(app)
    fig = zeros(256, 256);
    [X, Y] = meshgrid(1: 256);
    switch app.Wavetype
    case "Delta"
        fig(257 - app.position_y, app.position_x) = 1;
    case "Sine"
        an = app.angle_sin .* pi ./ 180;
        ph = app.phase .* pi ./ 180;
        fig = cos(2 .* pi .* app.frequency .* (cos(an) .* X + sin(an) .* Y)
+ ph);
    case "Rectangle"
        % have to edit the logic of rotate
        y_min = ceil(app.center_x - app.length ./ 2);
        y_max = ceil(app.center_x + app.length ./ 2);
        x_min = ceil(app.center_y - app.width ./ 2);
        x_max = ceil(app.center_y + app.width ./ 2);
        x_min = app.mapping1_256(x_min);
        x_max = app.mapping1_256(x_max);
        y_min = app.mapping1_256(y_min);
        y_max = app.mapping1_256(y_max);
        fig(x_min : x_max, y_min: y_max) = 1;
        fig = imrotate(fig, app.angle_rec, "bilinear", "crop");
```

```
    case "Gauss"
        fig = (X - 128.5) .^ 2 + (Y - 128.5) .^ 2;
        fig = exp(- fig ./ (2 .* app.variance)) ./ (2 .* pi .*
app.variance);
    case "Gabor"
        an = app.angle_gabor .* pi ./ 180;
        ph = app.phase_gabor .* pi ./ 180;
        x_an =  (X - 128.5) .* cos(an) + (Y - 128.5) .* sin(an);
        y_an = -(X - 128.5) .* sin(an) + (Y - 128.5) .* cos(an);
        fig = ...
            exp(-0.5 .* (x_an .^ 2 ./ app.variance_gabor + y_an .^ 2
./ app.variance_gabor)) ...
            .* cos(2 .* pi .* app.frequency_gabor .* x_an + ph);
    end
    app.Origin_fig_2 = fig;
end
```

首先通过不同波形的生成函数生成二维波形图像$app.Origin_fig_2$。

```
%% generate the DFT figure
function Ffigure = DFT_2(~, figure)
    [M, N] = size(figure);
    ux = (0 : M - 1)' * (0 : M - 1);
    vy = (0 : N - 1)' * (0 : N - 1);
    eMUX = exp(-2 * pi * 1i / M) .^ ux;
    eNVY = exp(-2 * pi * 1i / N) .^ vy;
    figure = figure + 0i;
    Ffigure = eMUX * figure * eNVY;
end
```

本段代码是进行DFT变换的核心代码，实现了一个对于二维$figure$图像的DFT变换。具体实现逻辑为先将$e^{-j2\pi(ux/M+vy/N)}$转化为$eMUX * eNVY$，而$eMUX, eNVY$的生成过程也是通过向量运算与矩阵运算实现的，因此整个运算过程速度可以得到保证。之后利用$eMUX * figure * eNVY$的公式就可以计算得到$figure$的DFT变换结果。

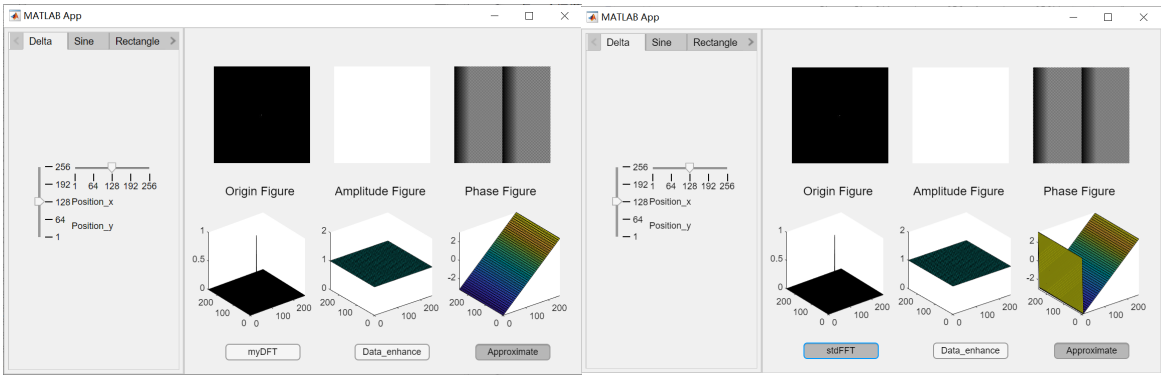
```
%% show the 3D surf
function show_3D(app)
    step = 6;% to present better
    [X, Y] = meshgrid(1 : 256, 1 : 256);
    [X_step, Y_step] = meshgrid(1: step: 256, 1 : step : 256);
    switch app.Wavetype
    case "Delta"
        surf(app.Axes_origin, X, Y, app.Origin_fig_2);
        surf(app.Axes_ampli, X_step, Y_step, app.Amplitude_fig_2(1 : step :
256, 1 : step : 256));
        surf(app.Axes_phase, X_step, Y_step, app.Phase_fig_2(1 : step : 256,
1 : step : 256));
    case {"Sine", "Rectangle", "Gauss", "Gabor"}
        surf(app.Axes_origin, X_step, Y_step, ...
        app.Origin_fig_2(1 : step : 256, 1 : step : 256));
        surf(app.Axes_ampli, X_step, Y_step, ...
        app.Amplitude_fig_2(1 : step : 256, 1 : step : 256));
        surf(app.Axes_phase, X_step, Y_step, ...
        app.Phase_fig_2(1 : step : 256, 1 : step : 256));
    end
end
```

```
%function to show the 6 figure
function fig_show(app)
    app.Wave_generate();
    if app.myDFTButton.Text == "myDFT"
        DFT_origin = app.DFT_shift(app.DFT_2(app.Origin_fig_2));
    elseif app.myDFTButton.Text == "stdFFT"
        DFT_origin = fftshift(fft2(double(app.Origin_fig_2)));
    end
    app.Amplitude_fig_2 = abs(DFT_origin);
    if app.Data_enhanceButton.Value == true
        app.Amplitude_fig_2 = log(app.Amplitude_fig_2 + 1);
    end
    if app.ApproximateButton.Value == true
        app.Amplitude_fig_2 = round(app.Amplitude_fig_2, 2);
    end
    app.Phase_fig_2 = angle(DFT_origin);
    app.Image_origin_2.ImageSource = cat(3, app.Origin_fig_2,
app.Origin_fig_2, app.Origin_fig_2);
    app.Image_ampli_2.ImageSource = cat(3, app.Amplitude_fig_2,
app.Amplitude_fig_2, app.Amplitude_fig_2);
    app.Image_phase_2.ImageSource = cat(3, app.Phase_fig_2, app.Phase_fig_2,
app.Phase_fig_2);
    app.show_3D();
end
```

本段代码主要是利用生成的波形图像以及DFT函数将计算得到的DFT图像并通过surf显示在坐标图中并将原图显示在图片位置，本段代码主要是为了实现“与标准FFT比较”， “对DFT结果进行数据增强”， “进行模糊/准确显示”三个功能进行了$if-else$判断。

另外就是为了使得surf图显示的图像足够颜色清晰，我对于DFT的显示结果采用了显示采样的方式，以避免surf在显示的过程中由于边界线的黑色覆盖了图像的彩色。

3.实验结果与分析



由上两图可知，在选择不同焦点时，可以看出对焦带来的前、背景模糊效果，并且拼接的图像边缘没有明显黑边与缝隙。

可以看到，在调整A并保持EV值相同后，可以与上左图比较发现明显景深提高了。，下面用自己的一组照片做与上述类似的对比。

上述两组三张图片的横向对比也可以发现，我拟合的滤波器大小函数，可以在两种不同大小($540\times360/1080\times768$)的图片上，均得到很好的效果，并且合成的效果也均很好。

4.遇到的困难与解决方法

本次实验遇到的最大困难就是第一部分提到的第一个难点，即合成图片后的黑边问题。在仔细观察了现象之后，我分析认为产生黑边的原因就是由于进行空域滤波后，边缘图像值与0进行了均值处理后造成的。在搜索了Matlab滤波器函数后，我发现了Matlab的imfilter函数可以使用 'replicate'参数，既可以扩展边界值，避免上述问题。

但我在尝试使用了上述函数之后，问题并没有得到解决，仔细思考了该函数的工作机理与我的图像矩阵输入后，我发现我所需要的前景图像边缘并不是滤波器得到的前景图像边缘。于是我自私的思考并观察了PNG图像进行imread后的矩阵数据，通过多种方法的实验与观察，我分析出了imread的两种读取PNG图像矩阵的结果是不同的。正如上文所述，在读取Alpha通道时，RGB通道的返回值是全面的原图，因此就可以利用原图信息得到边缘平顺的前景图片，之后进行图像合成。

5.收获

本次作业让我收获了Matlab的gui的编程方法，熟练掌握了Matlabapp的编程方法；同时我也熟练掌握了图片空域滤波的方法以及亮度的调节方法，对照片的三个参数有了更好的认识；同时，通过自己独立的解决了滤波后合成图像的黑边问题，我不仅帮助了很多其他同学解决同一问题，还提高了自己观察与分析解决问题的能力，极大的提高编程与解决问题的能力。

6.可能的改进方向

我认为我本次作业的完成质量还是比较可观的，有以下几个方面由于时间与能力的不足，我认为后续还可以加以提高：

1. 滤波器大小的拟合函数过于经验化，没有合适的理论依据与统一模型，后续可以进行理论分析得到更为合适的拟合公式
2. app编写的某些功能还是不够智能，可以进一步提高以完善用户体验。

7.参考文献

无