本科实验报告

课程名称：	机器感知实验
姓　　名：	金镇雄
学　　院：	元培
系：	智能科学系
专　　业：	智能科学与技术
年　　级：	19
学　　号：	1900094619
指导教师：	曲天书
职　　称：	副教授

2022 年　　3 月　　9 日

实验一、音频信号采集和分析

一、实验目的

熟悉音频信号的采集和音频信号的时域特征和频域特征。

二、实验要求

- 界面清晰美观
- 具有声音文件读入功能
- 具有声音播放功能
- 具有频谱分析功能
- 图形显示播放声音的时域波形和频域波形
- 实验结果分析
- 实验讨论

三、实验原理

- 使用 matlab 的 audioread()和 audiowrite()函数，读入和写出声音文件
- 声音时域、频域变换原理：fft();ifft()

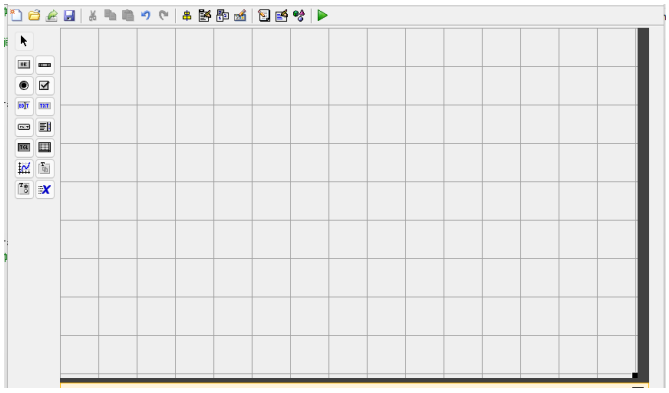
  时频分析：spectrogram()

四、主要仪器设备

PC 机、matlab、耳机

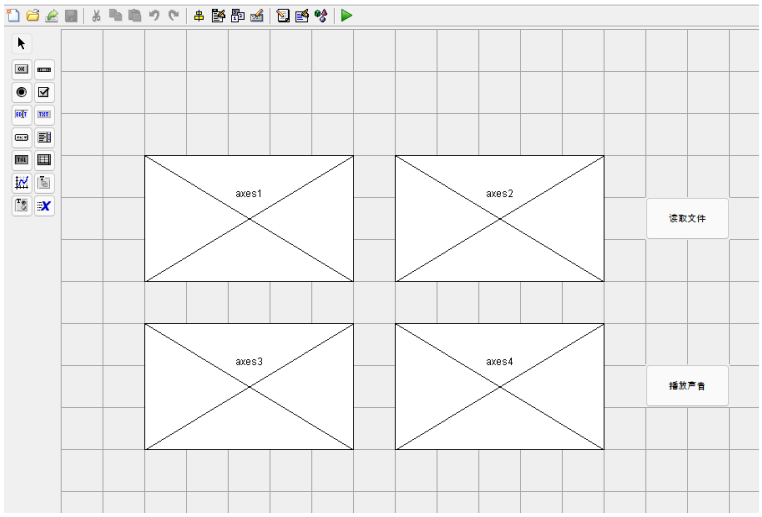
五、实验内容和步骤

1. 在 MATLAB 的命令窗口中键入 guide 打开如下 GUI 制作窗口。



2. 按照需求设计 GUI 界面并在窗体上添加各种控件。

本次实验中添加了四个坐标轴（Axes）和按钮（Push button）控件。四个坐标轴分别用于显示音频信号的整体时域波形、时频特性、实时信号波形以及实时频谱特性。第一个按钮（读取文件）用于选择并读入声音文件并显示其整体的时域和时频特性；第二个按钮（播放声音）用于播放声音并实时显示其波形以及频谱特性。



3. 读入声音文件

在 pushbutton1_Callback 函数中定义两个全局变量 audio 和 fs，分别是读入的声音信号以及其频率。用 uigetfile()和 audioread()函数选择并读入声音文件，将声音信息存储到两个全局变量中。

4. 显示声音信号的时域特性以及时频特性

在 GUI 界面的两个坐标轴 axes1 和 axes2 中分别显示时域特性和时频特性。时频特性分析用 spectrogram(x,windows,noverlap,nfft,fs)函数，其功能为实现短时傅里叶变换并得到信号的频谱图。其五个参数分别表示输入信号、窗口、重叠样本数、DFT 点数以及采样率。

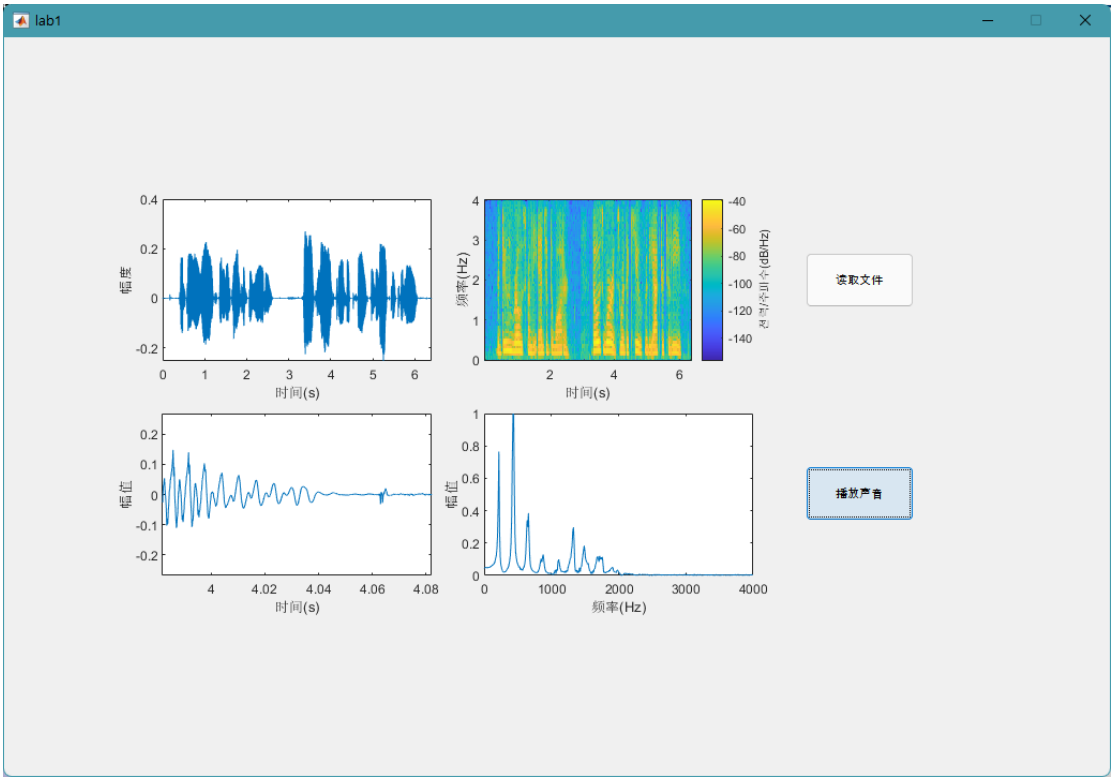
5. 播放声音并显示时域波形和频域波形

用 audioplayer()函数将音频信号转换为 audioplayer 对象，并以 play()函数播放声音。使用 audioplayer 对象的好处为通过其属性 CurrenctSample 能够访问当前播放的样本。播放声音的同时通过定时器按帧长（0.1s）重复执行显示当前时域和频谱波形的函数。

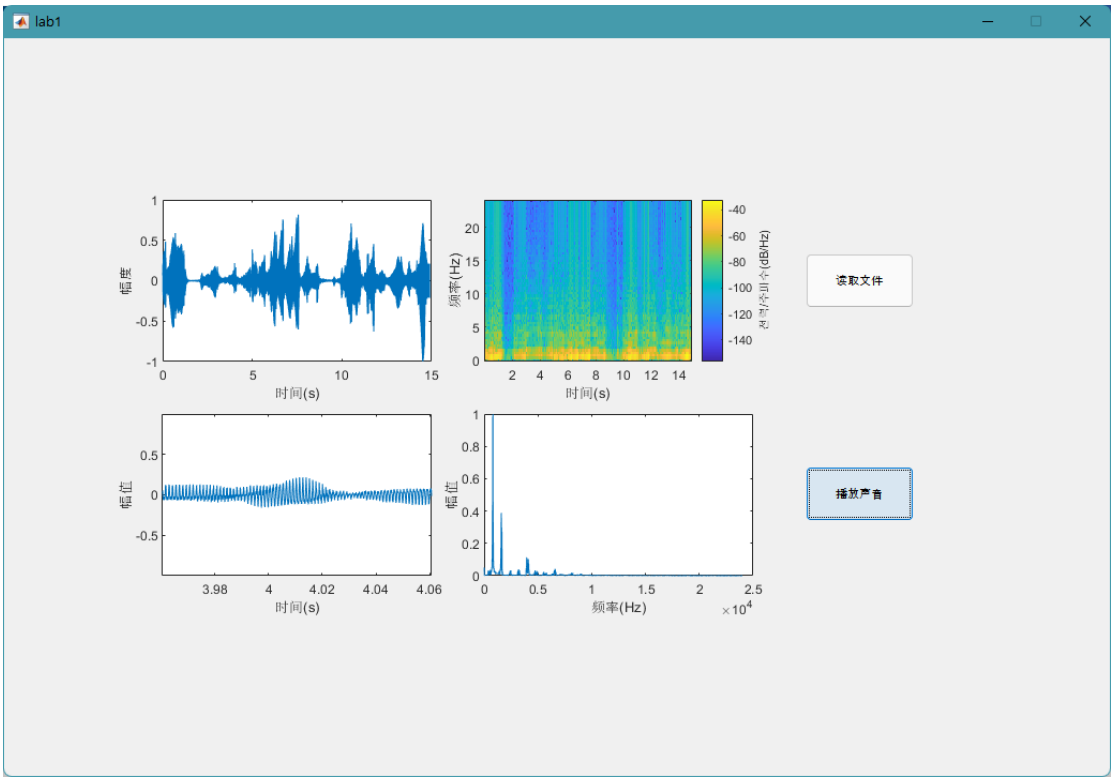
六、实验结果和分析

I. 实验结果

以 speech.wav 为音频样本



以 music.wav 为音频样本



II. 时频分析

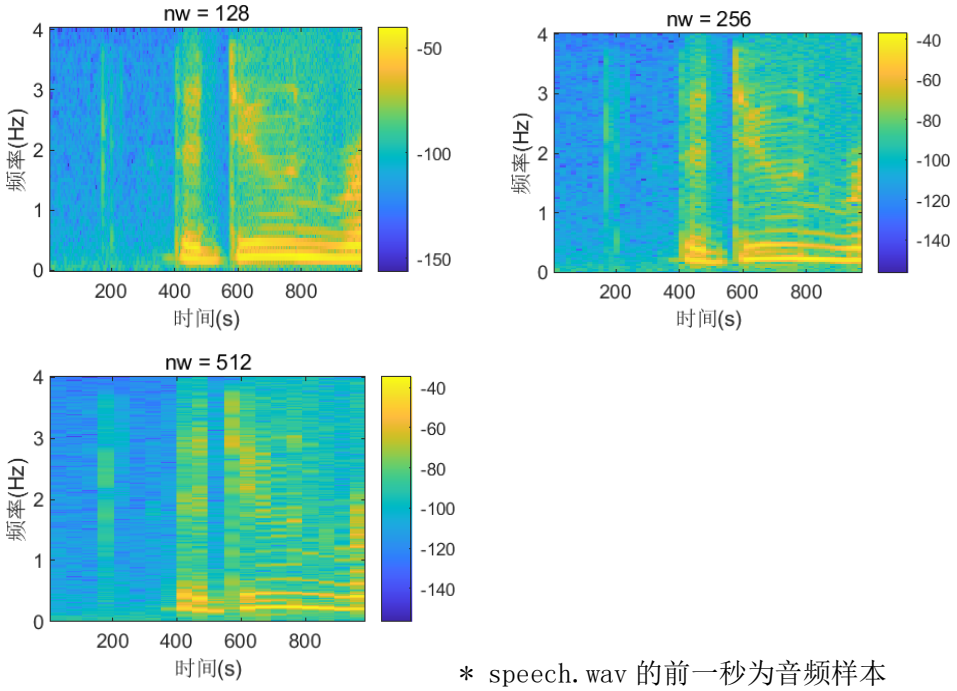
颜色：时频图的不同颜色区域表示对应频率下信号功率谱密度的分贝。

时间轴：其横轴为时间轴，即音频信号的长度。

频率轴：其竖轴为频率轴。因为信号经过傅里叶变换后频谱对称，实际信号频率范围是采样频率的一半，其显示范围亦然。

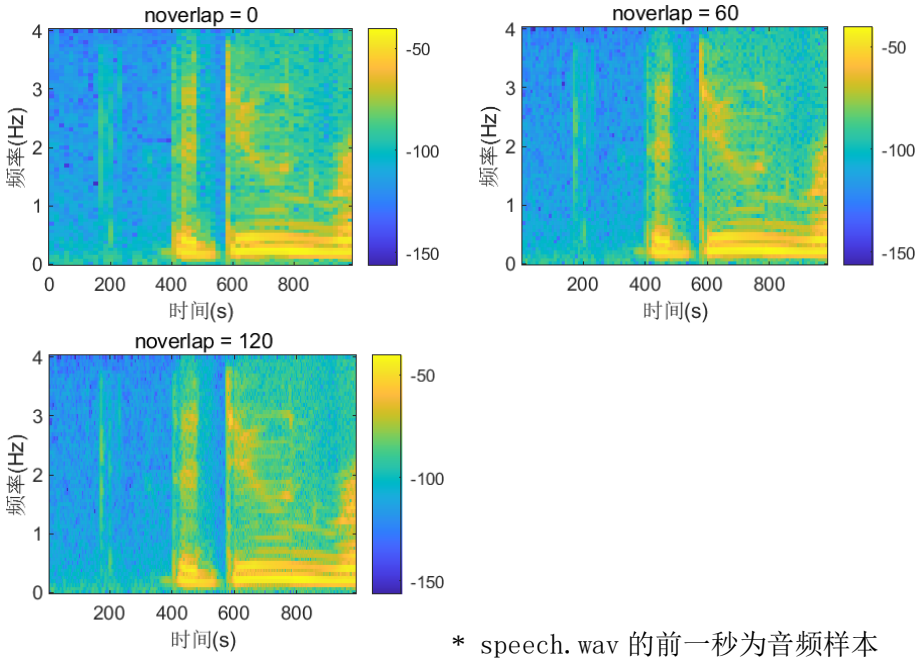
DFT 点数（nfft）：其值设置为最接近加窗信号长度的 2 的整数次幂。

窗口长度（nw）：窗口越长，频域分辨率越高，时间分辨率越低。如下三个图为各窗口长度下的时频图（noverlap=120）。很容易看出，随着窗口长度的增加，图像上表现为时间轴更加“粗疏”。这是因为每隔（Nw - noverlap）长度进行一次频率轴的更新，而 Nw 越大其长度越长，更新的间隔随之增大。另外，也可看出窗口长度越大，频域分辨率随之增高，频率轴上更加“细腻”。



* speech.wav 的前一秒为音频样本

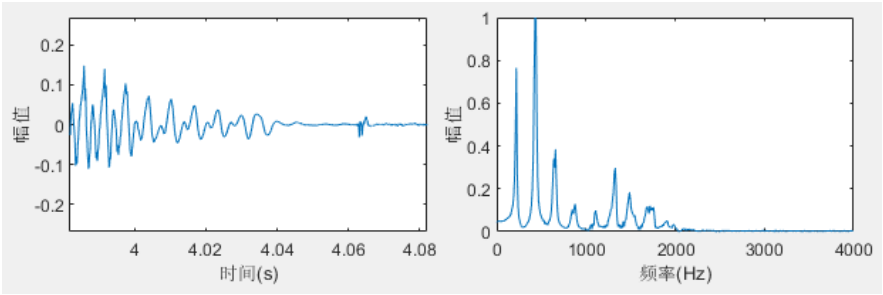
重叠样本数（noverlap）：表示窗口在移动的过程中与前一个窗口位置的重叠区域的大小，其值不能超过窗口长度。与窗口长度相反，重叠样本数越大，时间分辨率会降低。显然，重叠样本数越大，（Nw - noverlap）会减小，频率轴更新的间隔会减小，图像上表现为时间轴更加“细腻”。如下三个图为在窗口长度为 128 的条件下不同重叠样本数对应的时频图。



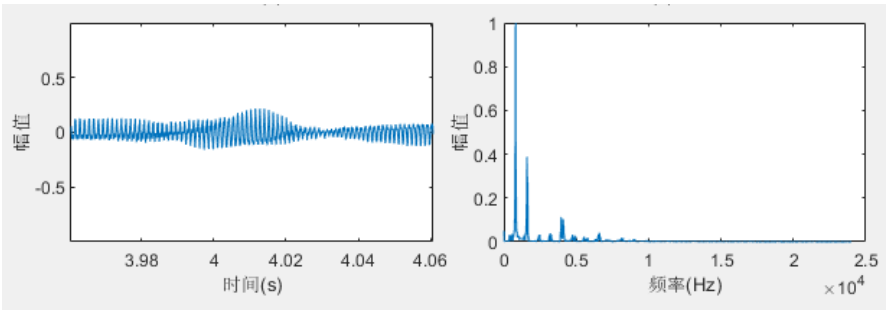
* speech.wav 的前一秒为音频样本

III. 实时时域波形和频域波形

以 speech.wav 为音频样本



以 music.wav 为音频样本



可看出有效地显示当前播放样本的时域和频域波形。因为信号经过傅里叶变换后频谱对称，实际信号频率范围是采样频率的一半，所以所化频谱图的横坐标范围为[0:fs/2]。随着时间的变化当前频谱的最大值也会发生变化，频域图的纵坐标会不断变化。因此在本频域分析中进行了归一化，将其最大值限定为 1。

七、讨论

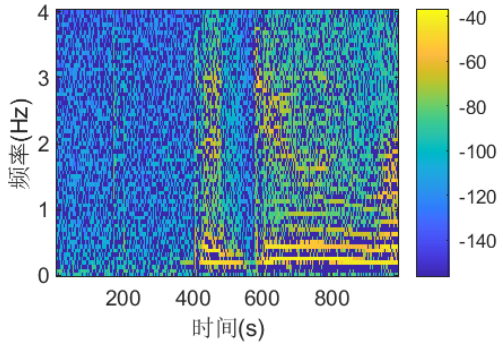
I. 如何提高频率分辨率

A. 提高窗口长度

如上所述窗口长达越长，其频率分辨率会增高。但要注意其时间分辨率会减小，因此，过于提高窗口长度的方法不值得采取。

B. 利用功率谱

利用功率谱，将能量集中的原理，提高频率分辨率。在 spectrogram 函数中增加‘reassigned’选项，就能实现此方法。下图为窗口长度为 128，重叠样本数为 120 时的时频图。可以看到频率分辨率与上图相比有了较大提升，但存在许多噪音。



II. 窗口长度与分辨率

由实验结果分析课发现，窗口长度与时间分辨率和频域分辨率有紧密的关系。窗口长度大，时间分辨率低，频率分辨率高；窗口长度小，时间分辨率高，频率分辨率低。若窗口大到整个信号长度时 STFT 退化成普通的傅里叶变换，无法时域分析；若窗口小到单个采样点时，无法做频域分析。因此，对于 STFT，最重要的是窗口长度的选取。

III. 小波问题

上述窗口长度的问题引出了小波变换等方法。 由于 STFT 采用的的滑动窗函数一经选定就固定不变，故决定了其时频分辨率固定不变，不具备自适应能力，而小波分析很好的解决了这个问题。小波变换克服了 STFT 窗口长度不随频率变化的缺点，而提供了一个随频率改变的“时间—频率”窗口，能自动适应时频信号分析的要求，从而可聚焦到信号的任意细节。

八、代码

```
function varargout = lab1(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @lab1_OpeningFcn, ...
                   'gui_OutputFcn',  @lab1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before lab1 is made visible.
function lab1_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for lab1
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = lab1_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
global audio;
global fs;
[file,~] = uigetfile();           % 选择音频样本
[audio,fs] = audioread(file);   % 读入音频信息
audio = audio(:,1);               % 若有两个以上的信道，则选择其中的第一道

axes(handles.axes1)               % 时域特性
```

```
t = (0:length(audio)-1)/fs;        % 时间轴
plot(t,audio);
xlabel('时间(s)'); ylabel('幅度');

axes(handles.axes2)                                          % 时频特性
nw = 128;                                                    % 窗口长度
window = hamming(nw);                                        % 窗口
noverlap = 120;                                              % 重叠样本数
nfft = 2^nextpow2(length(window));                           % DFT点数
spectrogram(audio,window,noverlap,nfft,fs,'yaxis');          % 显示时频图
xlabel('时间(s)'); ylabel('频率(Hz)'); % title('noverlap = 120');


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
global audio;
global fs;

player = audioplayer(audio,fs);        % 将音频信号转换成audioplayer对象
frame_len = 0.1;                       % 帧长为0.1s
ylim = max(abs(audio));                % 信号绝对值之最大值，用于确定时域的幅度范围
half_frame = round(frame_len*fs/2);    % 用于表示正在处理的帧的时间范围

pause(frame_len); play(player);        % 停顿剃短时间后播放声音

% 定义定时器
% StartDelay值为0，启动时没有延时
% TimerFcn为定时器被触发时执行的函数，这里为audio_plot()，显示信号的时域和频谱波形
% BusyMode为drop属性表示，当定时器需要执行TimerFcn，但前一次的TimerFcn仍然在执行时，
不执行当前TimerFcn
% ExecutionMode为fixedSpacing表示定时器的触发方式为循环触发且其间隔为前后两次被加入到
执行语句队列时刻之间
% Period为TimerFcn的执行周期，这里为帧长frame_len
my_Timer = timer('StartDelay',0, 'BusyMode','drop','Period',frame_len,
'ExecutionMode','fixedSpacing','TimerFcn',@(~,~)audio_plot(player,half_fram
e,ylim,handles));

start(my_Timer)                % 启动定时器
pause(length(audio)/fs+2)      % 等待至声音播放结束
delete(my_Timer)               % 删除定时器


function audio_plot(player,half_frame,ylim,handles)
global audio;
```

```
global fs;

% 计算走右侧索引
l = max(1,player.CurrentSample - half_frame);
r = min(length(audio),player.CurrentSample + half_frame);
frame = audio(l:r);
t = ([l:r]-1)/fs;
if l==1
    frame = [zeros(1,r-l+1),frame];
    t = [(2*l-r-1:l-1)/fs,t];
end
if r==length(audio)
    frame = [frame,zeros(1,r-l+1)];
    t = [t,(r+1:2*r-l+1)/fs];
end

axes(handles.axes3) % 当前时域波形
plot(t,frame); set(gca,'xlim',[t(1) t(end)],'ylim',[-ylim ylim]);
xlabel('时间(s)'); ylabel('幅值');

axes(handles.axes4) % 当前频谱波形
len = length(frame);                    % 信号长度
mag = abs(fftshift(fft(frame)));        % 频谱幅度为|F(e^jw)|
mag = mag/max(mag);                     % 归一化
f = [0:len/2]*fs/len;                   % 所画频谱横坐标范围为[0:fs/2]
plot(f,mag(round(len/2):len));
xlabel('频率(Hz)'); ylabel('幅值');
```