**根据音频拨号音，识别出当前所拨号码**

**——总结报告**

题 目 根据音频拨音号，识别出当前所拨号码

姓 名 应玲玲

学 号 3180432002

专业班级 电子信息工程181

所在学院 信息学院

指导教师（职称） 王一刚

二○一九 年 十一 月 十七 日

**目录**

一、 课题概述 .......................................................................................3

1、目的 ………........................................................................................3

2、原理 ……...........................................................................................3

3、实现 ……...........................................................................................3

二、 原理 ………........................................................................................4

1、DTMF（双音多频） ..........................................................................4

2、FFT(快速傅里叶变换) .........................................................................5

3、语音信号短时能量和过零率的特征.........................................................6

三、 程序代码及实现..............................................................................8

1、单个按键音识别 .................................................................................8

2、连续按键音识别................................................................................10

3、GUI设计（Pyqt5）.............................................................................14

四、 课题结果分析及结论....................................................................17

1、结果分析……………………………...........................................................17

2、结论……………………………..…………………………………………………………….17

**第一节 课题概述**

1、目的

（1）通过人为的按键输入形成录音m4a文件，再经Au处理形成wav文件，最终通过GUI界面输出结果。

（2）通过对连续语音信号的端点检测，了解语音截取的算法

2、原理

（1）每个按键音都是通过两个定的单音频率信号的组合来代表数字或功能。分为高频群和低频群，每个群里有四个不同的频率的信号，一共可以代表16个按键

（2）通过fft（快速傅里叶变换换）算法，把每个按键音时域信号转换成频域信号，再通过DTMF（双音多频）表进行分辨

3、实现

正确识别不同的拨号音。利用Python自带的wave库直接处理音频文件，通过matplotlib第三方包画出时域信号、频域信号、能量信号，实现对电话拨号音系统的简单的实验仿真。学习DTMF（双音多频）相关知识，知道双音多频的信号是用两个定的单音频率信号的组合来代表数字或功能。

主要涉及到电话拨号音识别的基本原理和识别的主要方法。利用python软件中的scipy模块中的fft（快速傅里叶变换换）算法、ifft（傅里叶反变换）算法以及带通滤波进行抗干扰实现对电话通信系统中拨号音的识别。并利用pyqt5、QTDesigner制作GUI界面。

**第二节 原理**

## DTMF（双音多频）

基本概念电话拨号有两种，脉冲和音频，所谓音频也成双音多频（DTMF）信号的拨号方式即是电话拨号时每按一个键，有两个音频频率叠加成一个双音频信号，十六个按键由八个音频频率区分。[双音多频](https://baike.baidu.com/item/%E5%8F%8C%E9%9F%B3%E5%A4%9A%E9%A2%91)的拨号键盘是4×4的[矩阵](https://baike.baidu.com/item/%E7%9F%A9%E9%98%B5" \t "_blank)，每一行代表一个低频，每一列代表一个高频。每按一个键就发送一个高频和低频的正弦信号组合，比如’1’相当于697和1209赫兹(Hz)。具体DTMF编制规则如表1所示。

表1 DTMF键盘

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 低频群  （HZ） | 高频群（HZ） | | | |
| 1209 | 1336 | 1447 | 1633 |
| 697 | 1 | 2 | 3 | A |
| 770 | 4 | 5 | 6 | B |
| 852 | 7 | 8 | 9 | C |
| 941 | \* | 0 | # | D |

每个按键对应的DTMF信号为，其中为低频单音，为高频单音。

通过用48000Hz对DTMF信号进行采样，采样后得到时域离散信号为：

根据采样定律大于被采样频率的两倍以上，满足要求。

再通过幅度量化得到二进制文件（wav文件）。

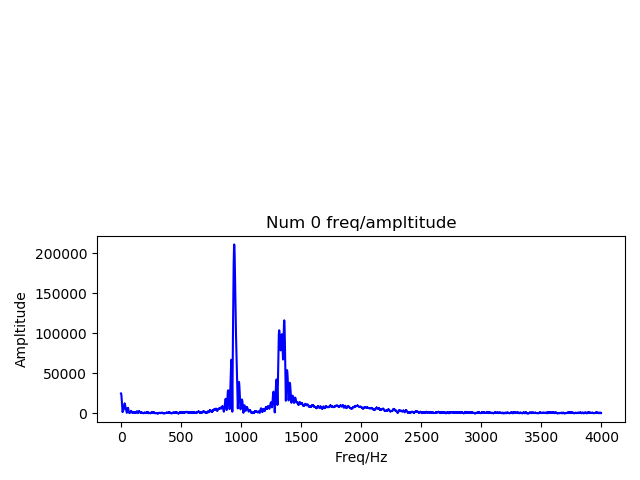
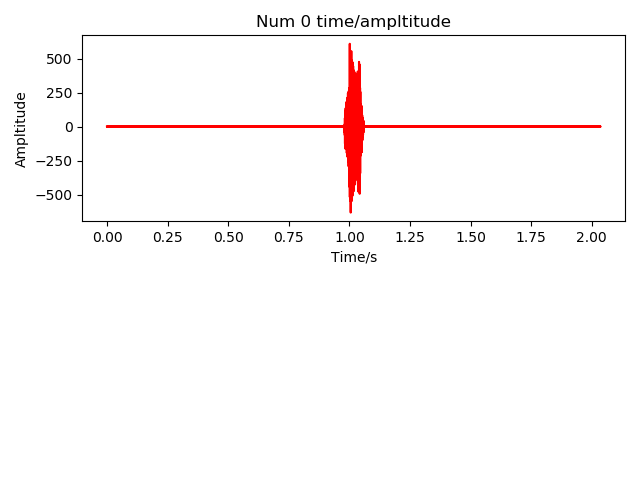
## 2 、FFT(快速傅里叶变换)

## 快速傅里叶变换 (fast Fourier transform), 即利用计算机计算离散傅里叶变换（DFT)的高效、快速计算方法的统称，简称FFT。快速傅里叶变换是1965年由J.W.库利和T.W.图基提出的。采用这种[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/209025" \t "_blank)能使计算机计算[离散傅里叶变换](https://baike.baidu.com/item/%E7%A6%BB%E6%95%A3%E5%82%85%E9%87%8C%E5%8F%B6%E5%8F%98%E6%8D%A2/6379901)所需要的[乘法](https://baike.baidu.com/item/%E4%B9%98%E6%B3%95/11036001)次数大为减少，特别是被变换的抽样点数N越多，FFT算法计算量的节省就越显著。FFT的基本思想是把原始的N点序列，依次分解成一系列的短序列。充分利用DFT计算式中指数因子 所具有的对称性质和周期性质，进而求出这些短序列相应的DFT并进行适当组合，达到删除重复计算，减少乘法运算和简化结构的目的。此后，在这思想基础上又开发了高基和分裂基等快速算法，随着数字技术的高速发展，1976年出现建立在数论和多项式理论基础上的维诺格勒傅里叶变换算法(WFTA）和素因子傅里叶变换算法。它们的共同特点是，当N是素数时，可以将DFT算转化为求循环卷积，从而更进一步减少乘法次数，提高速度。

在实验中使用python中第三方包scipy库（包含致力于科学计算中常见问题的各个工具箱。它的不同子模块相应于不同的应用。像插值，积分，优化，图像处理，统计，特殊函数等等）中的FFT函数完成傅里叶变换得到频域模型。

自变量的公式为：

其中framerate为采样频率，nframes为采样点数。

如按键“0”通过fftc()函数把时域信号，转换成频率信号（见图）：

（注：上图按键“0”的音频已经由Au静音转化处理过）

由于考虑到不需要高频的成分，所以选择4000做阈值，超出这个区域的都舍掉。

## 语音信号短时能量和过零率的特征

语音一般分为无声段，清音段和浊音段。一般把浊音认为是一个以基音周期为周期的斜三角脉冲串，把清音模拟成随机白噪声。由于语音信号是一个非平稳态过程，不能用处理平稳信号的信号处理技术对其进行分析处理。但由于语音信号本身的特点，在10-30ms的短时间范围内，其特性可以看作是一个准稳态过程，即具有短时性。因此采用短时能量和过零率来对语音进行端点检测是可行的。

本实验中通过每帧255个采样点数进行分帧。

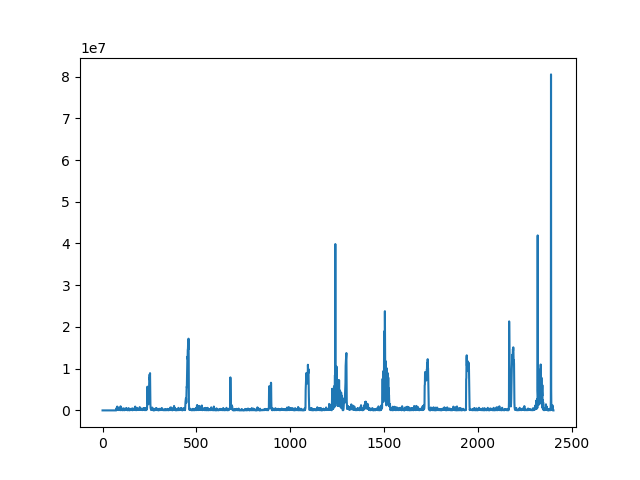
通过窗函数：

因为以255个采样点数进行分帧所以N=255

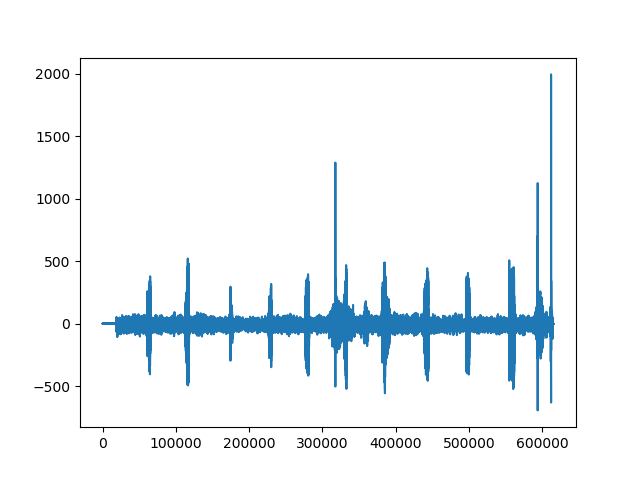
由语音短时能量的定义，就可以得到短时能量的计算公式：

即能量信号的计算公式

例如按键‘2698430719’通过分帧后进行后短时能量计算得到图1，时域信号见图2



**能量信号**

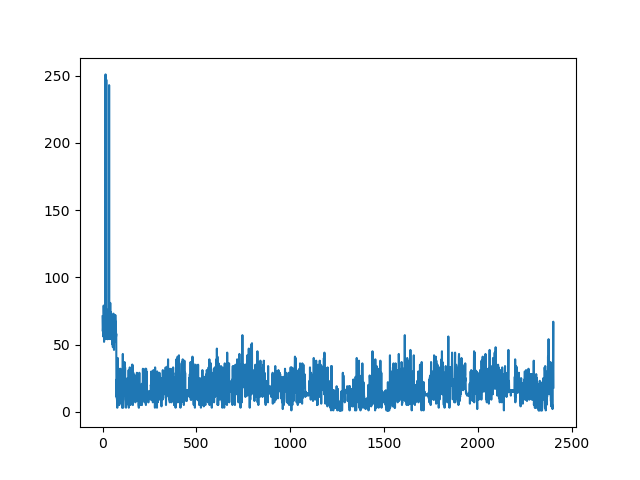


**时域信号**

再通过短时过零率计算公式：

得：

**短时过零信号**



1. **程序代码及实现**
2. 单个按键音识别

(1) **fttc**

# path为文件路径

# 输出为x频率信号的自变量，yf2是频率数据，framrate为采样率，nframes为采样点数，yt为单声道的时域数据

def fftc(path):

# 打开wav文件 ，open返回一个的是一个Wave\_read类的实例，通过调用它的方法读取WAV文件的格式和数据。

file = wave.open(path, "rb")

# 一次性返回所有的WAV文件的格式信息，它返回的是一个组元(tuple)：声道数, 量化位数（byte单位）, 采

# 样频率, 采样点数, 压缩类型, 压缩类型的描述。

information = file.getparams()

nchannels, sampwidth, framerate, nframes = information[:4]

# 读取声音数据，传递一个参数指定需要读取的长度（以取样点为单位）

str\_data = file.readframes(nframes)

file.close()

# 将波形数据转换成数组

# 需要根据声道数和量化单位，将读取的二进制数据转换为一个可以计算的数组

wave\_data = np.fromstring(str\_data, dtype=np.short)

# 将wave\_data数组改为2列，行数自动匹配。在修改shape的属性时，需使得数组的总长度不变。

wave\_data.shape = -1, 2

wave\_data = wave\_data.T

yt = wave\_data[0]

#单通道的时候不需要上述过程

yf2 = yt

x = np.arange(0, len(yf2) / 2) \* framerate / nframes

yf1 = abs(fft(yf2)) / len(yt) # 归一化处理

yf2 = yf1[range(int(len(yf2) / 2))] # 由于对称性，只取一半区间

return x, yf2, framerate, nframes, yt

**(2)** catch\_number

此函数是识别单个数字的按键音，输入为频域信号，采样频率，采样点数。输出为按键音所代表的数字。此函数在连续识别时候被调用。

def catch\_number(y, framerate, nframes):

frh = []

frl = []

math\_list = [] # 存放频率

# DTMF编码

fruquent = [697, 770, 852, 941, 1209, 1336, 1477, 1633]

math = {(697, 1209): '1', (697, 1336): '2', (697, 1477): '3', (697, 1633): 'A',

(770, 1209): '4', (770, 1336): '5', (770, 1477): '6', (770, 1633): 'B',

(852, 1209): '7', (852, 1336): '8', (852, 1477): '9', (852, 1633): 'C',

(941, 1209): '\*', (941, 1336): '0', (941, 1477): '#', (941, 1633): 'D'}

for fr in fruquent:

sum1 = sum(y[fr\* nframes //framerate-(20\* nframes //framerate):fr\* nframes //framerate])

sum2 = sum(y[fr\* nframes //framerate:fr \* nframes //framerate+(20\* nframes //framerate)])

sumx = sum1 + sum2

if fr < 1000:

frl.append([fr, sumx])

else:

frh.append([fr, sumx])

frl = sorted(frl, key=lambda x: x[1])

frh = sorted(frh, key=lambda x: x[1])

frl = frl.pop()

frh = frh.pop()

if frh[1] < 10:

return None

if frl[1] < 10:

return None

math\_list.append(frl[0])

math\_list.append(frh[0])

math\_list = tuple(math\_list)

for key, value in math.items():

if key == math\_list:

return value

1. 连续按键音识别
2. **calZero和calEnergy**

这两个函数是为了计算出能量信号和短时过零率给函数Endpoint\_detection提供数据。

#计算短时过零率

def calZero(waveData):

frameSize = 256

overLap = 0

wlen = len(waveData)

step = frameSize - overLap

frameNum = math.ceil(wlen/step)

zcr = np.zeros((frameNum,1))

for i in range(frameNum):

curFrame = waveData[np.arange(i\*step, min(i\*step+frameSize,wlen))]

curFrame = curFrame - np.mean(curFrame) # zero-justified

zcr[i] = sum(curFrame[0:-1]\*curFrame[1::] <= 0)

return zcr

#计算短时能量

def calEnergy(wave\_data) :

energy = []

sum = 0

for i in range(len(wave\_data)) :

sum = sum + (int(wave\_data[i]) \*\* ui.horizontalSlider\_2.value() )

if (i + 1) % 256 == 0 :

energy.append(sum)

sum = 0

elif i == len(wave\_data) - 1 :

energy.append(sum)

return energy

1. **Endpoint\_detection**

此两个函数是为了计算出检测出端点，输出为检测到的端点（因为以256为一帧，所以要\*256）。较高的短时能量作为阈值MH，利用这个阈值，我们就可以先分出语音中的浊音部分。取一个较低的能量阈值ML，利用这个阈值，我们可以从向两端进行搜索，将较低能量段的语音部分也加入到语音段，进一步扩大语音段范围。再通过短时过零率分辨。在最后加上语音持续时间分别，把小于10帧的非语音干扰信号删除。

def Endpoint\_detection(wave\_data, energy, Zero) :

sum = 0

energyAverage = 0

for en in energy :

sum = sum + en

energyAverage = sum / len(energy)

sum = 0

for en in energy[:10] :

sum = sum + en

ML = sum / 10

MH = energyAverage /4 #较高的能量阈值

ML = (ML + MH) / 2 #较低的能量阈值

sum = 0

for zcr in Zero[:100] :

sum = float(sum) + zcr

Zs = sum / 100 #过零率阈值

A = []

B = []

C = []

# 首先利用较大能量阈值 MH 进行初步检测

flag = 0

for i in range(len(energy)):

if len(A) == 0 and flag == 0 and energy[i] > MH :

A.append(i)

flag = 1

elif flag == 0 and energy[i] > MH and i - 21 > A[len(A) - 1]:

A.append(i)

flag = 1

elif flag == 0 and energy[i] > MH and i - 21 <= A[len(A) - 1]:

A = A[:len(A) - 1]

flag = 1

if flag == 1 and energy[i] < MH :

A.append(i)

flag = 0

# 利用较小能量阈值 ML 进行第二步能量检测

for j in range(len(A)) :

i = A[j]

if j % 2 == 1 :

while i < len(energy) and energy[i] > ML :

i = i + 1

B.append(i)

else :

while i > 0 and energy[i] > ML :

i = i - 1

B.append(i)

# 利用过零率进行最后一步检测

for j in range(len(B)) :

i = B[j]

if j % 2 == 1 :

while i < len(Zero) and Zero[i] >= 3 \* Zs :

i = i + 1

C.append(i)

else :

while i > 0 and Zero[i] >= 3 \* Zs :

i = i - 1

C.append(i)

# 除去帧数小于10的语音段

for i, data in enumerate(C):

if i % 2 == 1:

x = C[i] - C[i-1]

if x < 10:

del C[i]

del C[i-1]

print("过零率阈值，最终语音分段C:" + str(C))

count = []

for data in C:

count.append(data \* 256)

return count

（3）**continue\_math**

连续调用catch\_number，输出数字（以列表的形式输出）。

#打印出按键音数字  
def continue\_math(count,yt,framerate):  
 x = []  
 math = []  
 for i, data in enumerate(count):  
 x.append(data)  
 if (i + 1) % 2 == 0:  
 y1 = yt[x[0]:x[1]]  
 yf1 = abs(fft(y1)) / len(y1) # 归一化处理  
 yf2 = yf1[range(int(len(y1) / 2))] # 由于对称性，只取一半区间  
 math\_ = catch\_number(yf2, framerate, x[1] - x[0])  
 x = []  
 if math\_ == None:  
 continue  
 math.append(math\_)  
  
 return math

3、GUI设计（Pyqt5）

(1)pyqt5、 QTDesigner

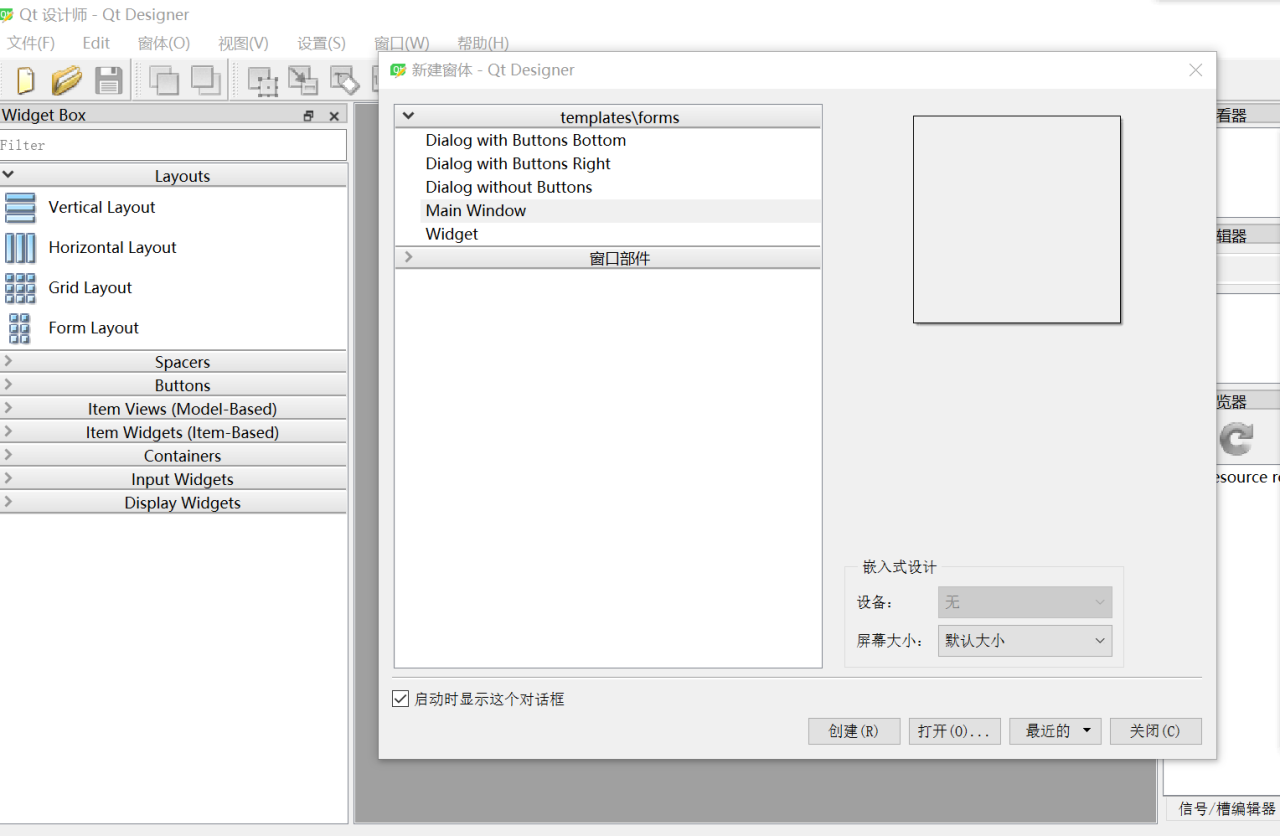
pyqt5：Qt是一套跨平台的C ++库，它们实现了高级API，可以访问现代桌面和移动系统的许多方面。其中包括定位和定位服务，多媒体，NFC和蓝牙连接，基于Chromium的Web浏览器以及传统的UI开发。

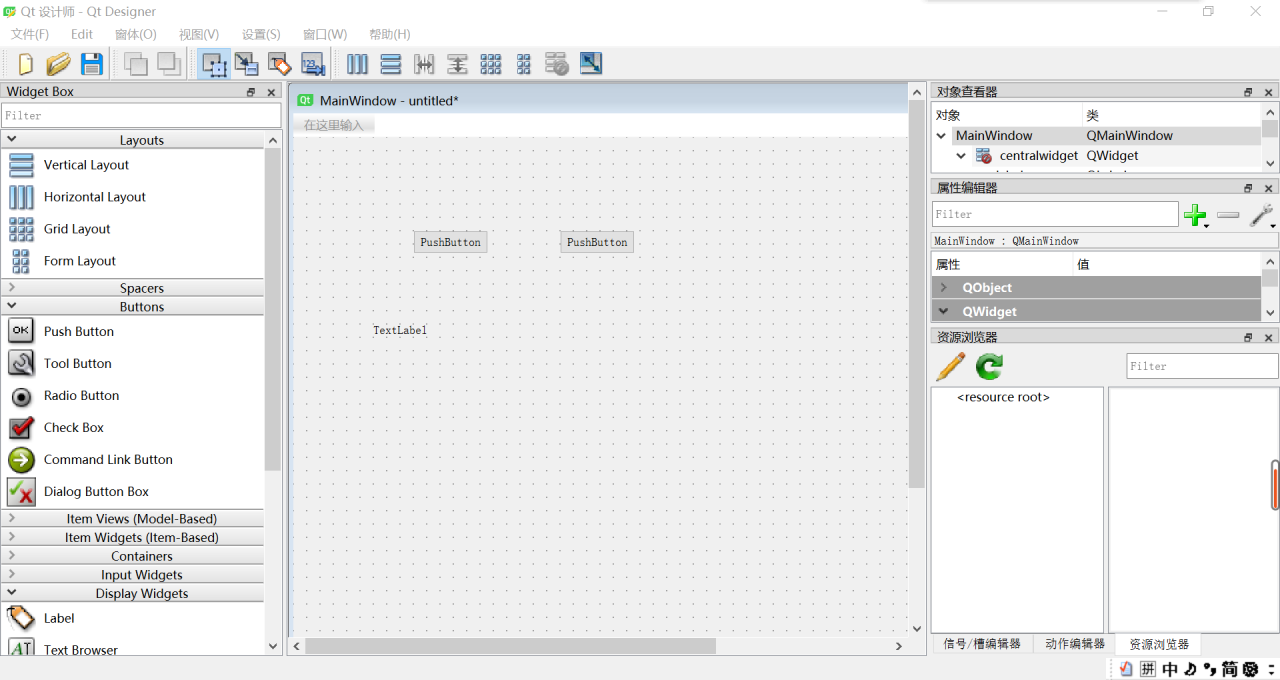
PyQt5是Qt v5的一套全面的Python绑定。它实现为超过35个扩展模块，并使Python能够在所有支持的平台（包括iOS和Android）上用作C ++的替代应用程序开发语言。

PyQt5也可以嵌入到基于C ++的应用程序中，以允许这些应用程序的用户配置或增强这些应用程序的功能。

QTDesigner：Qt Designer是Qt工具，用于使用Qt Widgets设计和构建图形用户界面（GUI）。可按照所见即所得（WYSIWYG）的方式编写和自定义窗口或对话框，并使用不同的样式和分辨率对其进行测试。

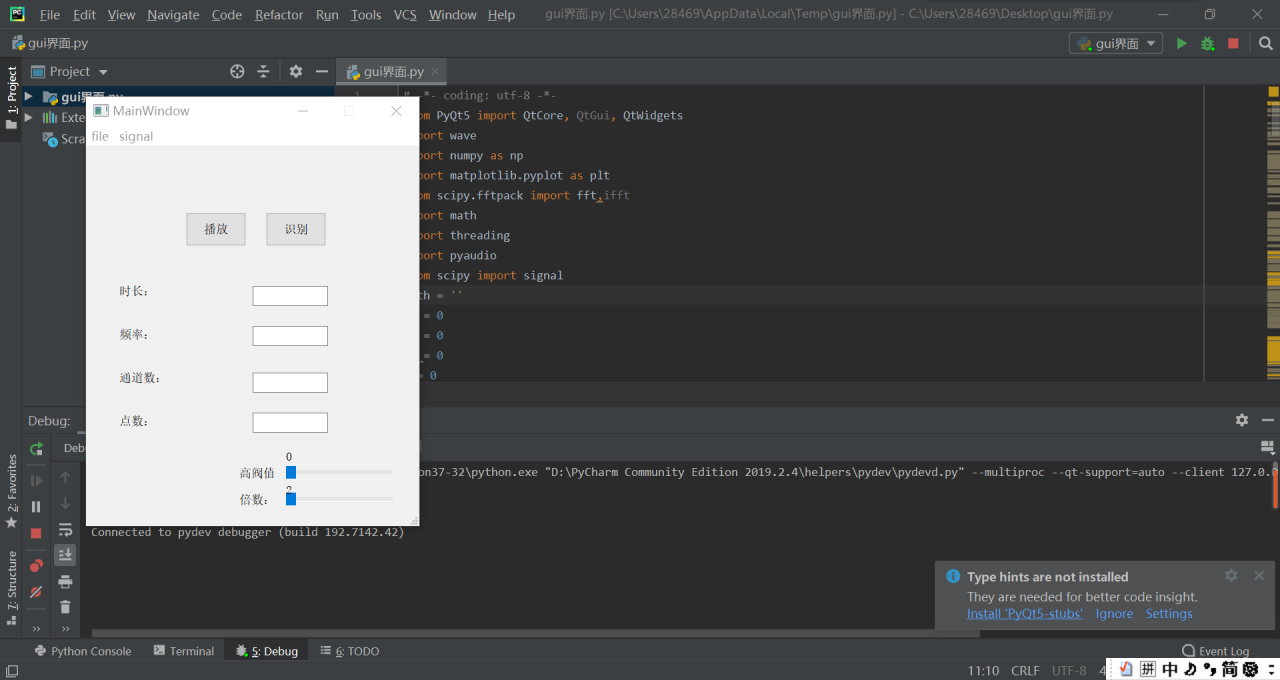
使用Qt信号和插槽机制，使用Qt Designer创建的小部件和表单与编程代码无缝集成，因此可以轻松地将行为分配给图形元素。Qt Designer中设置的所有属性都可以在代码中动态更改。此外，小部件升级和自定义插件等功能允许在Qt Designer中使用自己的组件。

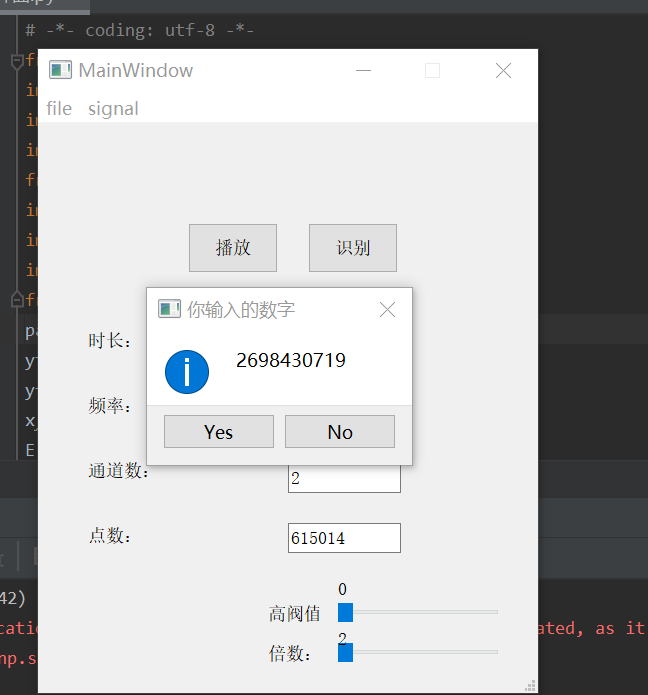




可以直接拖拽各类元件，实现界面设计。最后保存生成.Ui文件，通过cmd pyuic5 \*.ui -o \*.py 可以直接生成代码文件。

最后实现的gui界面：





**第四节 实验结果分析及结论**

1.1、实验结果

|  |  |
| --- | --- |
| 按键音个数 | 准确率% |
| 5 | 100% |
| 10 | 100% |
| 15 | 93% |
| 20 | 98% |

1.2、结果分析

（1）在音频文件较为理想的情况下，识别正确率高。一些杂音的存在易导致识别错误。

（2）在识别连续按键音时，若音频文件中各按键音的间隔时间过短，则识别错误率普遍较高。

2、结论

音频识别正确率较高。在识别连续按键音时，若音频文件中各按键音的间隔时间过短，则识别错误率普遍较高。在音频文件较为理想的情况下，识别正确率高。一些杂音的存在易导致识别错误。在打开GUI页面调用函数时，经常出现页面卡死现象，且识别速度时快时慢。