第5章 chapter 5

音频处理

在学习NAO的音频功能之前，首先了解语音信号的一些相关知识。

人具有“说”和“听”的生理机能，或者说具有语音生成和语音/音频识别功能。人的发声器官由喉、声道和嘴三部分组成，发声器官产生不同强度的气流，控制喉中的声带产生不同频率和不同幅度的振动，产生声波。正常人的发音频率范围为100～5000Hz。人的听觉器官为耳。声音(空气振动)传入耳道引起鼓膜振动，经内耳把机械运动变换为神经信号，最后传给大脑，识别出不同的语音含义。

NAO也能够实现“说”和“听”，与人进行交流、会话，本章介绍NAO的语音识别和语音生成功能。

5.1 音频数据

语音识别和语音生成都涉及对音频数据的表示、存储、建模和处理。

5.1.1 存储音频

利用话筒获取到的声音信号是模拟、连续的电信号，将这些电信号存储到计算机中需要经过采样、量化和编码三个处理过程。

1. 采样

由于不能记录一段时间音频信号的所有值，只能使用采集样本方法记录其中的一部分。采样是对模拟信号周期性地记录信号大小的方法。图5.1显示了从模拟信号上选择10个样本代表实际的声音信号。

声音信号变化越快，单位时间中采集的样本数就需要越多，才能还原出原始信号。在图5.1中，最后一段声音信号变化较快，按照图示的采样频率，已经不能有效恢复原始信号了。依据采样定理(在一个信号周期中，至少需要采两次样，才能有效恢复原始信号)，采样频率至少是声音最高频率的2倍。采样频率越高，还原的信号越接近于原始信号。

NAO提供的采样频率包括8000、11025、12000、16000、22000和24000Hz(赫兹)。

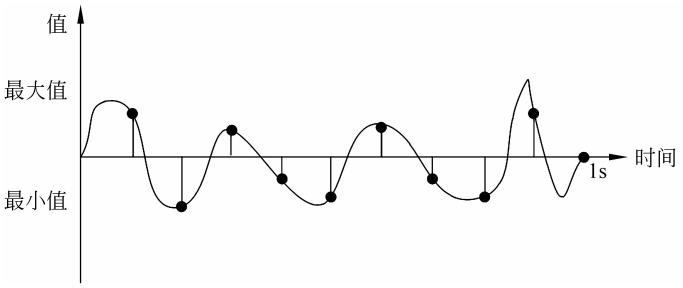

图 5.1　一个音频信号的采样(1 秒钟内采集 10 个样本值)

根据处理音频信号的频率范围,可以做相应选择。例如,在做语音识别时,由于语音信号最高频率只有几千 Hz,采样频率可以选择 8000Hz,8000Hz 也是电话系统使用的语音采样频率。

2. 量化

从每次采样得到的测量值是真实的数字。量化指的是将样本的值截取为最接近的整数值的过程。例如,实际值为 17.2,截取为 17;实际值为 17.8,截取为 18。

3. 编码

量化后的数值需要被编码成位模式。NAO 使用 16 位二进制数表示这些数字。编码二进制位数越多,数值表示就越精确。

4. 存储

NAO 在存储录制的声音时,可以保存为以下两种不同的格式。

WAV 格式:采样的音频数据不做其他处理,仅仅加上一个文件头,说明采样频率、通道个数、编码长度等信息,存储的文件就是常见的 WAVE 文件。WAV 格式文件由于没有压缩数据,占用的存储空间特别大。

OGG 格式:采用 Vorbis 压缩算法对采样音频数据进行压缩处理而得到的文件,文件扩展名为.ogg。所用算法是开源的、无专利费用。

采样频率为 8000Hz,每秒产生的音频数据为 8000×16b=128 000b=16 000B。

采样频率为 22kHz,每秒产生的音频数据约为 44KB,1min 将产生大约 2.5MB 的音频数据,在双声道采样时将达到每分钟产生 5MB 音频数据。

5.1.2　ALAudioRecorder

NAO 头部安装了 4 个话筒,频率范围是 150Hz～12kHz,每个话筒对应一个声道,具有独立的采样和编码电路。

NAOqi 提供的录音模块为 ALAudioRecorder,保存文件格式为 WAV 或 OGG。

NAO 可以录制如下声音:

(1) 四声道,48000Hz,OGG;

(2) 四声道,48000Hz,WAV;

(3) 单声道(frontright, frontleft, rearleft ,rearright), 16000Hz,OGG;

(4) 单声道(frontright, frontleft, rearleft ,rearright), 16000Hz,WAV。

使用 ALAudioRecorder()方法如下。

(1) startMicrophonesRecording()开始录音,非阻塞调用方法。

格式:

```
startMicrophonesRecording(filename,type,samplerate,channels)
```

其中,filename 为保存录音文件的绝对路径;type 为录音的格式,也可以为 WAV 或 OGG 格式;samplerate 为采样频率;channels 为请求所用通道,元组类型。

(2) stopMicrophonesRecording()停止录音。

代码清单 5-1 录音

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.record =ALProxy("ALAudioRecorder")
    def onLoad(self):
        pass
    def onUnload(self):
        pass
    def onInput_onStart(self):
        record_path ='/home/nao/record.wav'
        self.record.startMicrophonesRecording(record_path, 'wav', 16000, (0,0,1,0))
        time.sleep(10)
        self.record.stopMicrophonesRecording()
        self.onStopped()
        pass
    def onInput_onStop(self):
        self.onUnload()
        self.onStopped()
```

程序运行后,将使用 C 话筒录制 10s 的声音,采样频率 16000Hz,以 WAV 格式存储在机器人/home/nao 目录下。

5.1.3 ALAudioPlayer

ALAudioPlayer 可以播放多种格式的音频文件,提供常见的播放功能(播放、停止、暂停、循环等)。ALAudioPlayer 生成的音频流最终驱动机器人的两个扬声器发声。ALAudioPlayer 提供了三十余种方法,控制音频文件的播放,表 5.1 列出了部分方法。

表 5.1 ALAudioPlayer 部分方法

方 法 名	说 明
playFile(filename)	播放指定文件。filename 为文件名。相当于执行 loadFile 和 play
getCurrentPosition(taskId)	返回当前播放位置(秒)。taskId 为非阻塞调用播放方法返回值
goTo(taskId, position)	跳到播放文件的指定位置。position 为跳转位置,实数
getFileLength(taskId)	获取当前播放文件的长度(秒)
loadFile(fileName)	预先装入声音文件,但不播放。返回值为播放文件的 taskId 值
play(taskId)	播放
play(taskId, volume, pan)	volume 为音量,取值范围为[0.0～1.0];pan 为立体声参数(－1.0:左声道/1.0:右声道)
playFileFromPosition(fileName, position, volume, pan)	指定位置播放

播放音频文件前,利用 winscp 工具,将待播放文件上传到机器人/home/nao 目录下。ALAudioRecorder 只能在实体机器人上运行,ALAudioPlayer 可以在虚拟机器人上运行。但在虚拟机器人上 ALAudioPlayer 只能播放 WAV 和 OGG 格式的文件。

代码清单 5-2 播放并获取当前位置(播放上例录制文件)

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.aup=ALProxy("ALAudioPlayer")
    def onLoad(self):
        pass
    def onUnload(self):
        pass
    def onInput_onStart(self):
        fileId =self.aup.post.playFile("/home/nao/record.wav")
        time.sleep(5)
        currentPos =self.aup.getCurrentPosition(fileId)
        self.logger.info(str(currentPos))
        pass
    def onInput_onStop(self):
        self.onUnload()
        self.onStopped()
```

声音播放是阻塞调用,可以使用模块代理的 post 对象,在并行线程中创建任务。每个 post 调用产生一个任务 ID 值。本例中 playFile 在播放文件时产生一个 ID 值,在执行 getCurrentPosition()方法时作为参数,读取当前文件播放位置。

打开日志查看器窗口,程序运行后可以看到输出结果为 6.0。

5.1.4 音频特征

1. 语音时域特征

声音信号是随时间变化的连续信号。如图 5.2 所示是 NAO 说“开始”时的声音信号。

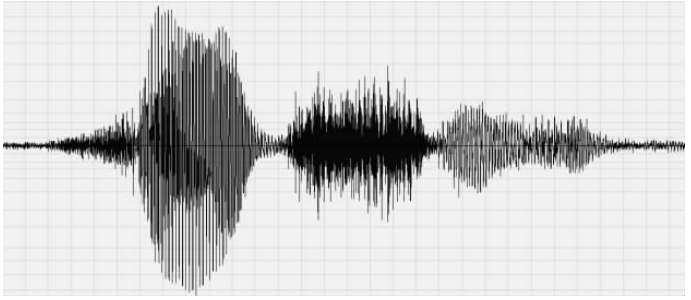

图 5.2 NAO 说“开始”的音频信号

一般情况下,发音时每个字对应一段声波,各个字之间会有一定的间隔时间,甚至一个字之间也有一定的间隔时间。图中信号前半部分振幅较大的部分为“开”的声音信号,后半部分为“始”的声音信号。

2. 傅里叶分析

法国数学家傅里叶证明:任何正常的周期为 T 的函数 $g(t)$,都可以由(无限个)正弦和余弦函数合成:

$$g(t) = a_0 + \sum_{n=1}^{\infty}(a_n\cos(2\pi nft) + b_n\sin(2\pi nft))$$

此处 $f=1/T$ 是基频,a_n 和 b_n 是正弦和余弦函数的 n 次谐波的振幅,这种分解叫傅里叶级数。通过傅里叶级数可以重新合成原始函数。各项系数可以通过变换公式求出。

图 5.3 所示为二进制信号 1110001 的前 4 次谐波合成结果。从图中可以看出,使用的谐波数量越多,叠加的结果就越接近于原始函数。图中的前三列横坐标为时间,最后一列横坐标为频率,纵坐标为幅度的平方根(a_n 与 b_n 平方和开平方)。使用基频的 n 次谐波的振幅数据(频域),可以重塑原始信号(时域)。

利用傅里叶变换,可以将随时间变化的语音信号(时域信号),转换为一组与频率相关的振幅(频域信号),进而研究信号的频谱结构和变化规律。

3. 离散傅里叶变换

离散傅里叶变换(DFT)是信号在时域和频域上都是离散的。DFT 运算结果也是将时域信号的采样值变换为频域不同谐波的振幅。

在实际应用中通常采用快速傅里叶变换(FFT)计算 DFT。FFT 采用了一些近似计算方法,误差很小,运算效率高于 DFT,在 C 和 Python 语言中都有实现函数。通常情况

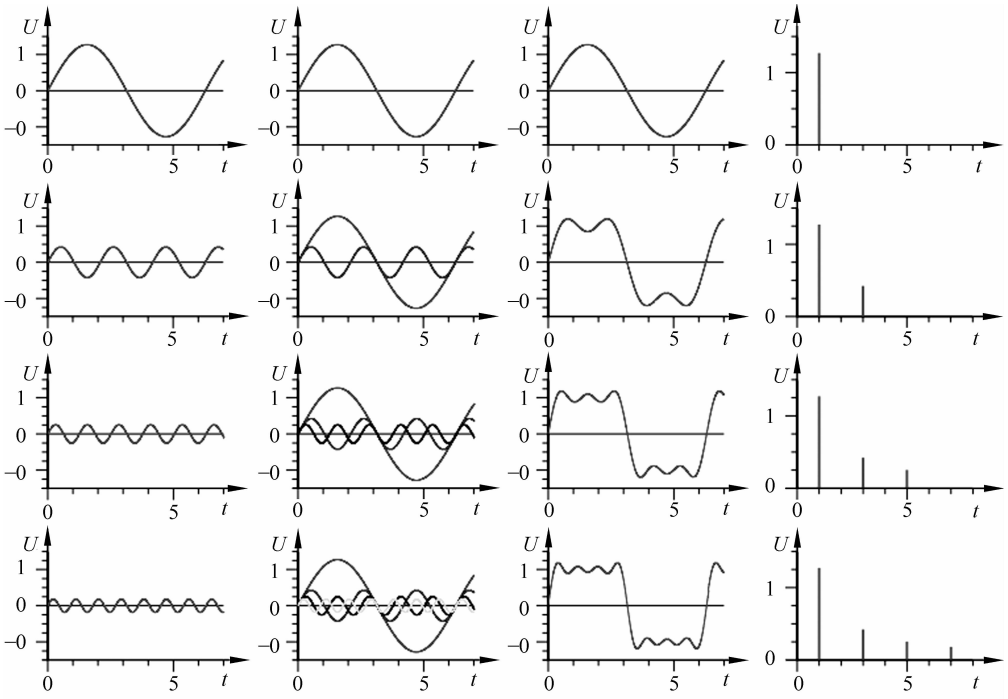

图 5.3 二进制信号 1110001 的前 4 次谐波合成结果

下,FFT 变换两端(时域和频域上)的序列是有限长的(如取 512 或 1024 个数值)。也就是说,一个包含 512 个元素的数组,输入 FFT 算法,输出也是 512 个元素的数组。输入数据表示的是随时间变化的信号采样值,输出数据表示的是这段音频信号在各个频率上的幅度。

4. 音频特征

在任意一个自动语音识别系统中,第一步就是提取特征。换句话说,需要把音频信号中具有辨识性的成分提取出来,然后把其他的信息扔掉,如背景等。

在语音识别技术中,最常用到的语音特征就是梅尔倒谱系数(MFCC)。计算梅尔倒谱系数需要经过预加重,分帧、加窗、快速傅里叶变换,Mel 滤波器组,计算每个滤波器组输出的对数能量,离散余弦变换等过程。

Python 语音处理库 Librosa 提供了 MFCC 的计算方法。

5.2 ALAudioDevice

ALAudioDevice 管理音频的输入和输出,提供了一组操作音频输入设备(话筒)和输出设备(扬声器)的 API。其他的音频模块(ALAudioPlayer 除外)做输入和输出时都使用 ALAudioDevice 提供的 API,如图 5.4 所示。

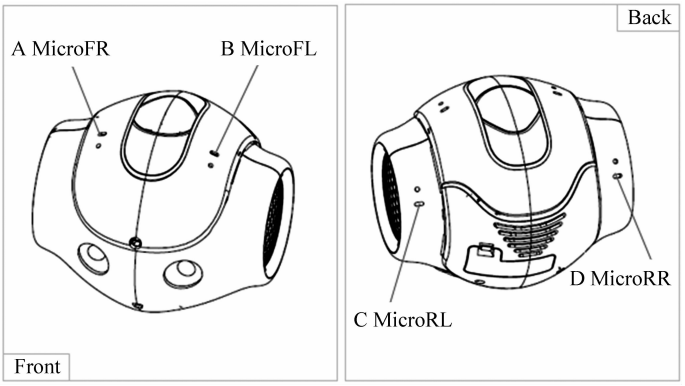

图 5.4 麦克风位置

ALAudioDevice 基于标准的 Linux ALSA(Linux 声音库)库,通过声音驱动程序,与话筒和扬声器通信。

ALAudioDevice 与其他模块关系如图 5.5 所示。

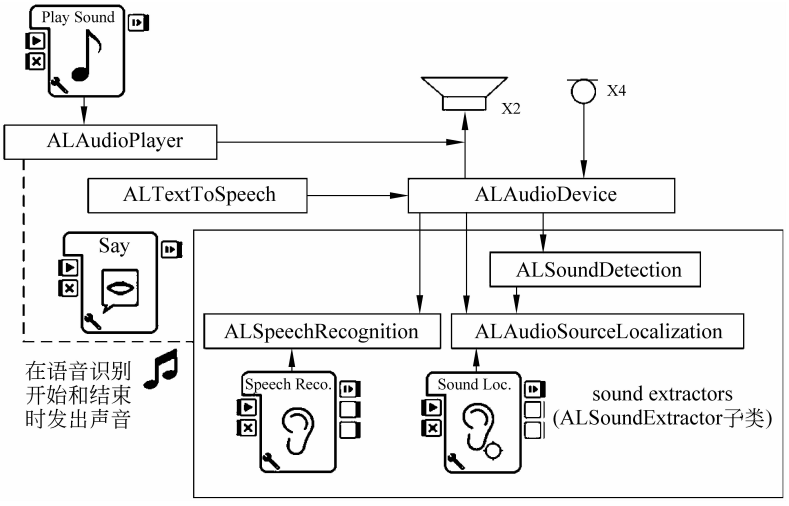

图 5.5 ALAudioDevice 与其他模块关系

5.2.1 输出

1. 数据格式

ALAudioDevice 可以通过以下帧速率之一将数据发送到扬声器:

两声道,交错数据(声道 1 数据,声道 2 数据,声道 1 数据,……),16000Hz(设置亚洲语言时默认);

两声道,交错数据,22050Hz(默认设置非亚洲语言时);

两声道,交错数据,44100Hz;

两声道，交错数据，48000Hz。

2. 输出方法

(1) sendLocalBufferToOutput(nbOfFrames, buffer)，本地模块(运行在机器人上的模块)将存放在数据缓冲区声音数据发送到扬声器。数据格式为 16b 两声道交错数据，缓冲区长度不能超过 16 384B。

其中，nbOfFrames 为缓冲区中包括的两声道声音数据帧数。在两声道交错数据格式中，1 帧长度为两声道的数据长度，4B；buffer 为发送缓冲区内存地址。

发送成功方法返回值为真，否则返回值为假。

(2) sendRemoteBufferToOutput(nbOfFrames, buffer)，远程模块（运行在机器人之外的模块)将存放在数据缓冲区声音数据发送到扬声器。

任何 NAOqi 模块都可以在需要时调用适当的方法向 NAO 的扬声器发送数据。如果数据格式正确，不必做其他配置就可以调用。

3. 输出相关方法

(1) setParameter(parameter，value)，设置输出采样率。其中，parameter 只能设置为"outputSampleRate"，value 为设置的输出采样率，可以设置为 16000, 22050, 44100 或 48000。

(2) setOutputVolume(volume)，设置系统的输出音量。其中，volume 取值为[0，100]。

系统音量可以由 getOutputVolume()方法获取。

代码清单 5-3　播放 WAV 文件(文件内容发送给扬声器)

```
import math
import os.path
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.aad=ALProxy("ALAudioDevice")
    def onLoad(self):
        pass
    def onUnload(self):
        pass
    def onInput_onStart(self):
        self.aad.setParameter("outputSampleRate",16000)
        songPath ="/home/nao/hongdou.wav"  #播放前上传文件，hongdou.wav 为两声
                                           #道，采样频率为 16 000Hz
        size=os.path.getsize(songPath)
        nframe=(size-44)/4        #帧数，WAV 头部为 44B，每帧 4B(两声道，每声道 16b)
        with open(songPath,"rb") as file:
            file.seek(0)
            data=file.read()
```

```
        data=data[44:]                          #头部以后为数据部分
        self.aad.sendLocalBufferToOutput(int(nframe), id(data))
                                                #使用 id()函数取数据在内存地址
        pass
    def onInput_onStop(self):
        self.onUnload()
        self.onStopped()
```

代码清单 5-4 生成正弦波并播放

```
import numpy as np
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.aad=ALProxy("ALAudioDevice")
    def onLoad(self):
        pass
    def onUnload(self):
        pass
    def onInput_onStart(self):
        sineTableSize =1024
        outputBufferSize =16384
        sine=np.arange(sineTableSize,dtype=np.int16)
        left_phase =0
        right_phase =0
        nbOfOutputChannels=2
        sampleRate =16000
        freq =1000
        duration =5.0
        fOutputBuffer=np.arange(nbOfOutputChannels * outputBufferSize,dtype
                      =np.int16)
        for i in range(0,sineTableSize):
          sine[i]= int (32767 * math.sin((float(i)/float(sineTableSize)) *
                    math.pi * 2.0))
        ratio=1.0 /(1.0 / float(freq) * float(sampleRate)/float(sineTableSize))
        inc=int(math.ceil(ratio))
        nbOfBuffersToSend =int(math.ceil(duration * sampleRate/
                           outputBufferSize))
        for d in range(0,nbOfBuffersToSend):
            i=0
            while i<nbOfOutputChannels * outputBufferSize:
                fOutputBuffer[i] =sine[left_phase]
                fOutputBuffer[i+1] =sine[right_phase]
                left_phase =left_phase+inc
                if left_phase >=sineTableSize:
```

```
                left_phase = left_phase - sineTableSize
            right_phase = right_phase + inc
            if right_phase >= sineTableSize :
                right_phase = right_phase - sineTableSize
            i = i + nbOfOutputChannels
        buffer = fOutputBuffer.tostring()
        self.aad.sendLocalBufferToOutput(outputBufferSize, id(buffer))
    pass
def onInput_onStop(self):
    self.onUnload()
    self.onStopped()
```

5.2.2 自定义模块

NAOqi 模块都是库中的类。从 autoload.ini 加载库时，这些模块类将自动实例化。

除了使用 NAOqi 模块外，也可以自定义模块。自定义模块类的基础类是 ALModule。从 ALModule 派生的类，可以“绑定”方法。模块名称和方法将被通告给代理，这样其他模块就可以使用自定义模块了。

1. 模块分类

自定义模块可以是远程模块也可以是本地模块。如果是远程的，在机器人外部运行。远程模块可以从外部调试，但在速度和内存使用方面效率较低。如果是本地的，在机器人本地运行。本地模块比远程模块效率高。

每个模块都可以包含多种方法，某些方法可以限定为从模块外部调用。不管模块是远程的还是本地的，调用这些限定方法的方式是一样的。

本地模块是在同一进程中启动的两个（或更多）模块，本地模块由 NAOqi 作为代理（Broker），统一管理，可以共享变量，可以直接调用彼此的方法。在做一些闭环控制时，必须使用本地模块。

远程模块使用网络进行通信。远程模块需要代理与其他模块通信。Broker 负责所有网络通信。

2. 远程模块的连接

远程模块可以通过使用代理方法，将其代理连接到其他模块的代理上，实现与其他模块通信。连接方式如图 5.6 所示，可以是 Broker 到 Broker，也可以是 Proxy 到 Broker。

3. ALModule 方法

ALModule 模块类是自定义模块类的基类，负责为其子类通告方法。ALModule 方法如表 5.2 所示。