

# 基于 Android 平台的吉他智能校准系统设计与实现

于阿强,汪方军,孙 存,朱锡祥,吕 钊  
(安徽大学 计算机科学与技术学院,安徽 合肥 230601)

**摘 要:**传统吉他弦音校准主要通过人耳听觉来判断,不仅要消耗大量时间与精力,同时准确性也难以保证。为了解决这一问题,设计并建立了一种便携的基于 Android 平台的吉他智能校准系统。所建立系统在对原始弦音信号进行去噪、分帧、加窗等预处理后,通过短时能量与过零率进行端点检测,在此基础上,使用傅里叶变换计算端点检测后弦音信号的频率,然后与标准弦音频率进行比较,给出误差供调音者参考。在实验室环境下,以 44 100 Hz 为采样频率,对 10 把吉他进行了系统测试验证。测试验证结果表明,经校准后的吉他弦音频率跟标准频率对比的平均误差在 2.68% 以内,校准精度较为理想;所建立系统操作简单、方便,不仅可以适用于吉他调音,而且也适用于其他弦类乐器的校准。

**关键词:** 吉他智能校准;端点检测;快速傅里叶变换;JNI 技术

**中图分类号:** TP302

**文献标识码:** A

**文章编号:** 1673-629X(2017)07-0140-05

**doi:** 10.3969/j.issn.1673-629X.2017.07.032

## Design and Implementation of a Guitar Intelligent Calibration System with Android Platform

YU A-qiang, WANG Fang-jun, SUN Cun, ZHU Xi-xiang, LYU Zhao  
(College of Computer Science and Technology, Anhui University, Hefei 230601, China)

**Abstract:** The traditional way to tune a guitar is mainly through hearing, which is not only a waste of time and energy, but also a lack of accuracy. To solve this problem, a portable guitar intelligent tuning system has been designed and established based on Android platform, which uses endpoints detection through short-time energy and zero-crossing counts after the pretreatments including de-noising, framing and windowing of the original sound signal. On the basis of it, Fourier transform is used to calculate the sound signal frequency. Then error between the result and standard sound frequency has been provided available information for tuners. In the laboratory environment, with sampling frequency of 44 100 Hz, 10 guitars have been tested and verified systematically. Results show that compared with standard frequency, the average error of the calibrated guitars' sound frequency keeps within 2.68%, which is satisfying. The system is simple and convenient. It can apply to the tuning of guitar and other stringed instrument.

**Key words:** guitar intelligent calibration; endpoint detection; fast Fourier transform; JNI technology

## 0 引 言

调音作为吉他演奏的前期必要准备工作,直接决定了弹奏的音高是否准确。但是传统的吉他调音方法如六音笛法、听自然泛音法等,调音过程困难枯燥,而且无法保证准确度。虽可借助专业的调音硬件(如吉他电子液晶智能校准器),但此类产品成本太高,不易携带。近期出现了基于 Android 平台和 iOS 平台的吉他智能校准系统,但其计算速度较慢或准确度不高。

为解决上述问题,设计并建立了一种基于 Android 平台的便携式吉他智能校准系统。该系统在对原始弦音信号进行去噪、分帧、加窗等预处理后,通过短时能

量与过零率进行端点检测,使用傅里叶变换计算端点检测后弦音信号的频率,并通过 JNI 技术快速实现对弦音的检测、处理和基音频率的计算。实验结果表明,系统具有计算速度快、性能稳定、准确度高等优点。

## 1 系统与实现

基于 Android 平台的吉他智能校准系统由两部分组成:弦音处理模块和 Android 平台实现模块。其中,弦音处理模块包括弦音端点检测、频率计算及校准。Android 平台实现模块是在弦音处理模块的基础上实现基于 Android 平台的弦音录入、波形及结果显示等。

收稿日期:2016-05-24

修回日期:2016-09-08

网络出版时间:2017-04-28

基金项目:安徽省自然科学基金(1408085QF125);安徽大学大学生创新创业训练项目(201510357040)

作者简介:于阿强(1994-),男,研究方向为信息安全;吕 钊,博士,副教授,研究生导师,研究方向为智能信息处理与人机交互技术。

网络出版地址: <http://kns.cnki.net/kcms/detail/61.1450.TP.20170428.1702.002.html>

系统结构如图 1 所示。

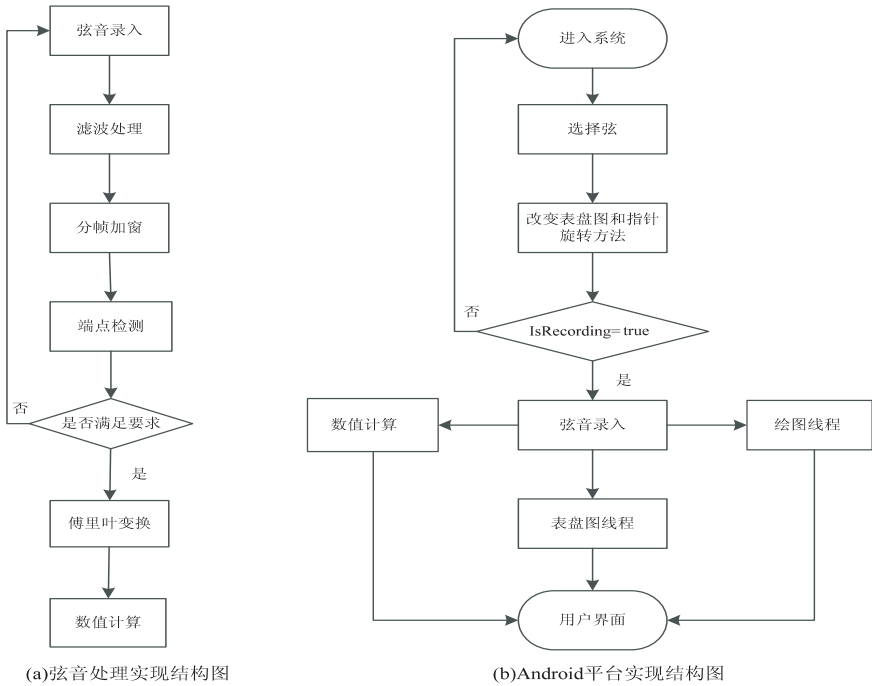


图 1 系统结构图

1.1 弦音处理

1.1.1 预处理

由于实际调音过程中会受到很多外界噪声的干扰,因此系统对经录音模块传入的声音数据进行滤波降噪处理,以提高弦音检测的准确性,最大限度地消除噪声对弦音的干扰,同时,吉他六根弦的频率范围在 82 ~ 330 Hz,所以系统使用一个三阶 50 Hz 高通滤波器<sup>[1]</sup>和一个三阶 350 Hz 低通滤波器级联而成的带通滤波器<sup>[2]</sup>对录入的弦音信号进行滤波处理。

1.1.2 分帧加窗

弦音信号虽然是时变的,但其在短时范围内的变化较小,因此可以认为其具有短时平稳性。所建立系统使用一个有限长的窗序列作为  $\{\omega(n)\}$  窗函数,并让其滑动截取一段段弦音信号进行处理,这一过程称为对弦音信号的分帧加窗<sup>[3]</sup>。为了保证相邻两帧之间的平稳过渡和弦音信号的连续性,一般要让相邻两帧之间重叠一部分,帧移便是后一帧对前一帧的位移量。因为采用汉明窗可使 99.963% 的能量集中在窗谱的主瓣内,而最大的旁瓣峰值小于主瓣峰值的 1%,所以所建立系统使用汉明窗作为窗函数<sup>[4]</sup>,其定义为:

$$\omega(n) = \begin{cases} 0.54 - 0.46\cos[\frac{2\pi n}{L-1}], & 0 \leq n \leq L-1 \\ 0, & n = \text{else} \end{cases} \quad (1)$$

其中,  $L$  为窗长。

1.1.3 端点检测

弦音信号的端点检测是所建立系统中非常重要的

一个环节,其不但可以减少处理时间,而且能排除无声段噪音的干扰,提高弦音的品质。端点检测的目的是将弦音信号从复杂的噪音信号中分离出来,并确定弦音的起始位置和终止位置<sup>[5]</sup>。所建立系统采用了基于短时能量和短时过零率的端点检测算法。

(1) 短时能量。

设弦音波形时域信号为  $x_n(m)$ , 加汉明窗  $\{\omega(n)\}$  分帧处理后,得到第  $n$  帧弦音信号  $x_n(m)$ , 且满足:

$$x_n(m) = \omega(m) * x((n-1) * \text{inc} + m), 1 \leq i \leq f_n \quad (2)$$

其中,  $m = 1, 2, \dots, L; n = 1, 2, \dots, f_n; L$  为帧长;  $\text{inc}$  为帧移长度;  $f_n$  为总帧数。

设第  $n$  帧弦音信号  $x_n(m)$  的短时能量用  $E(n)$  表示,则其计算公式为:

$$E(n) = \sum_{m=0}^{L-1} x_n^2(m), 1 \leq n \leq f_n \quad (3)$$

(2) 短时过零率。

短时过零率表示一帧弦音信号波形穿过横轴(零电平)的次数。弦音信号  $x_n(m)$  的短时过零率  $Z(n)$  定义为:

$$Z(n) = \frac{1}{2} \sum_{m=0}^{L-1} |\text{sgn}[x_n(m)] - \text{sgn}[x_n(m-1)]|, \quad 1 \leq n \leq f_n \quad (4)$$

其中,  $L$  为帧长;  $f_n$  为总帧数;  $\text{sgn}[\ ]$  为符号函数,即:

$$\text{sgn}[x] = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases} \quad (5)$$

将短时能量和短时过零率结合进行端点检测<sup>[6]</sup>,其过程如图 2 所示。

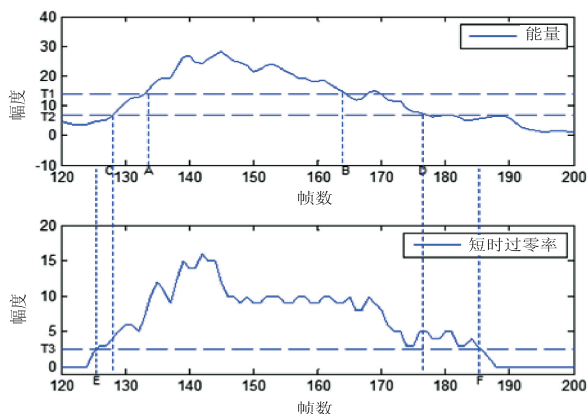


图 2 端点检测示意图

(3) 第一级判决。

①根据弦音短时能量的包络线选取一个较高的门限  $T_1$ , 找到交点  $A$  和  $B$ 。弦音起止点则位于门限  $T_1$  和短时能量包络交点所对应的时间点之外 (即  $AB$  段之外)。

②在平均能量上确定一个较低的门限  $T_2$ , 并从  $A$  点向左、 $B$  点向右搜寻, 找到交点  $C$  和  $D$ ,  $C$  和  $D$  两点便是双门限法根据短时能量所判定的弦音的起止点。

(4) 第二级判决。

根据背景噪音的短时平均过零率确定门限  $T_3$ , 然后以短时过零率为准, 从  $C$  点往左、 $D$  点往右搜寻, 找到短时过零率首次低于门限  $T_3$  的两点  $E$  和  $F$ , 这两点就是弦音的起止点。

图 3 给出了根据上述方法对第五根弦的弦音进行端点检测的结果, 图中左右两边的实线分别表示弦音的起始位置和终止位置。

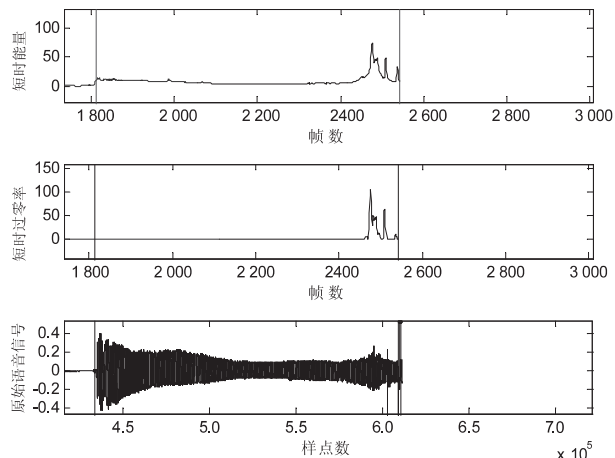


图 3 第五根弦的端点检测示意图

#### 1.1.4 快速傅里叶变换

为了获取弦音信号频率并提高计算效率和速度, 系统对端点检测后的结果进行快速傅里叶变换 (FFT)<sup>[7]</sup>。而快速傅里叶变换又是离散傅里叶变换的

快速算法<sup>[8]</sup>, 某一帧的离散傅里叶变换定义式如下:

$$X_n(e^{j\omega}) = \sum_{m=-\infty}^{+\infty} x(m)\omega(n-m)e^{-j\omega n} \quad (6)$$

其中,  $\omega(n-m)$  是窗函数。

通过对每一帧弦音信号进行离散傅里叶变换, 将其从时域转换到频域, 从而获得实际频率。

### 1.2 Android 平台实现

#### 1.2.1 录音调用

Android 中封装了许多实用的应用程序接口 (API), 使得系统对于硬件的调用十分便捷。AudioRecord 类是 Android 中进行音频数据录制和处理的关键类。因为相比于其他录音类, 其允许对获得的 PCM 数据进行直接处理, 所以所建立系统选用 AudioRecord 类中的方法获得外界音频数据。

Android 录音前, 需要定义一个最小缓冲区, 用于暂时存储音频数据, 声明最小的录音缓冲区需要下面三个关键参数<sup>[9]</sup>。

//设置音频采样率为 44 100 Hz 是为了获得更高的计算精度

```
static final int frequency = 44 100;
```

```
//设置双声道
```

```
static final int channelConfiguration = AudioFormat.CHANNEL_IN_STEREO;
```

```
//音频数据格式: 每个样本 16 位
```

```
static final int audioEncoding = AudioRormat.ENCODING_PCM_16BIT;
```

通过以上三个参数, 然后调 AudioRecord.getMinBufferSize 方法就能返回录音所需要的最小缓冲区大小。具体代码示例如下:

```
recBufsize = AudioRecord.getMinBufferSize(frequency, channelConfiguration, audioEncoding);
```

录音需要长时间的工作, 为了不造成系统卡顿, 为其单独开辟了新的线程。Android 提供了一个工具类 AsyncTask (异步执行任务)。AsyncTask 的功能就是开辟一个独立于主线程的线程<sup>[10]</sup>, 能够在后台进行一些耗时的计算, 从编程语法上显得优雅了许多, 不再需要子线程和 Handler 就可以完成异步操作并且刷新用户界面<sup>[11]</sup>。在所建立系统的设计中, RecordTask 继承自 AsyncTask 类, 在获取音频数据的同时将信息分别传输至主线程波形图绘制模块和频率数据计算线程。此部分的关键代码结构如下:

```
class RecordTask extends AsyncTask<Object, Object, Object> {
    .....
    //在后台运行的线程, 在这里进行录音、记录数据的操作
    protected Object doInBackground (Object...params) {
        try {
            audioRecord.startRecording(); //开始录制
            //音频数据的记录
```

```
.....  
//执行主线程 UI 刷新(波形图绘制)  
publishProgress();  
{  
catch(Throwable t){  
}  
return null;  
}  
}
```

在方法 `doInBackground` 中, `read(short[] audioData, int offsetInShorts, int sizeInShorts)` 函数的作用是将硬件获取到的数据读出,放入缓冲区。数据的处理和显示都是根据 `read` 函数读出的内容进行修改和处理。

### 1.2.2 波形图绘制模块

Android 提供了一个 `surfaceView` 的绘图控件, `surfaceView` 可以直接从内存或者 DMA 等硬件接口取得图像数据,在 Android 开发中是个非常重要的绘图容器<sup>[12]</sup>。此程序的波形图在一个实例化的 `surfaceView` 控件上绘制。在录音模块的 `RecordTask` 类中,调用了一个名称为 `publishProgress` 的方法,这个方法用于 UI 的更新。调用 `publishProgress` 方法时,系统会自动调用 `onProgressUpdate` 方法中的代码<sup>[13]</sup>,如下:

```
@Override  
protected void onProgressUpdate(Object...values){  
long time=new Date().getTime();  
if(time-c_time>=draw_time){  
ArrayList<Short> buf=new ArrayList<Short>();  
.....//省略部分操作,使得 buf 的大小和 SurfaceView 控件  
的宽度保持一致  
buf=(ArrayList<Short>)inBuf.clone();//从音频数据缓冲区  
中复制音频信息  
}  
//利用更新数据绘图函数  
SimpleDraw(buf,baseLine);  
c_time=new Date().getTime();  
super.onProgressUpdate(values);  
}
```

这段代码在方法的起点会获取当前的时间 `time` 并记录,在方法的结尾处会记录此刻时间 `c_time`,鉴于绘图功能的反复调用, `time-c_time` 获取的就是上一次绘图结束到这一次绘图的时间间隔,如果时间间隔小于预定的绘图间隔时间,则拒绝绘图,防止没有必要地占用过多的 CPU 资源。图 4 为绘图策略的解释图。Inbuf 是一个 `ArrayList` 类型的数组链表,当数据的宽度满足绘图要求时,将此数据复制到 `buf` 的数组链表中, `buf` 作为 `simpleDraw` 方法的参数,用于具体的波形图绘制。 `simpleDraw` 的主要原理就是描点、画线、利用 Android 自带的 `drawLine` 方法 `public void drawLine(float startX, float startY, float stopX, float stopY, Paint paint)` 进行图形绘制。

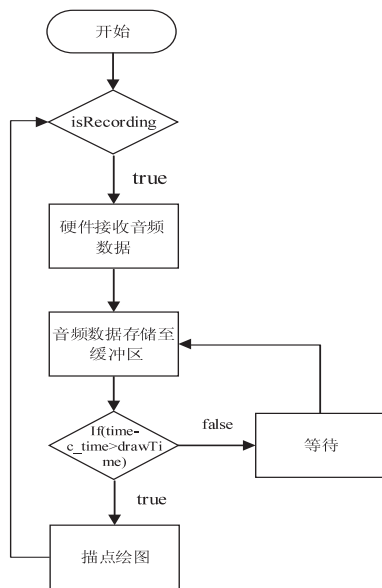


图 4 绘图策略解释图

实际程序中, `startX`、`startY` 初始值为 0,之后每一次都是上一次绘画中的 `stopX`、`stopY`,当 `X`、`Y` 超出画布范围时将归零。这个部分的主要代码是:

```
for(int i=0;i<buf.size();i++){  
y=buf.get(i)/rateY+baseLine;//调节缩小比例,调节基  
准线  
canvas.drawLine(start+(i-1)*divider,py,start+i*divider,  
y,mPaint);  
py=y;  
}
```

对 `buf` 中的每一个数据,都取出来描点,将这些点连接起来,在用户界面会显示连续的波形图,见图 5。

### 1.2.3 音频计算模块

在录制音频数据的线程中,开辟一个额外的线程 `ComputingThread`,这个线程会将缓冲区中的数据传递到音频计算模块中用于计算, `ComputingThread` 调用 `ComputingFrequency` 类文件中获取数据的方法,将缓冲区中的音频数据传递到 `ComputingFrequency` 类中, `ComputingFrequency` 会根据传递来的数据调用预处理模块的代码进行计算,并将计算后的频率数据返还。

### 1.2.4 频率指示器

频率指示器显示区别于波形显示,运用了图形的旋转,将计算的频率结果可视化。指针和表盘图是两个不同的图片,指针在初始状态位于表盘的正中间,这是使用了 Android 布局方法中的 `RelativeLayout` 实现的。取最左侧和最右侧的  $180^\circ$  空间为有效区域,根据吉他一共有六根弦的情况,需要根据每一条弦列出旋转函数,在此以 E1 音为例加以说明。E1 音的范围是  $63.2 \sim 100$  Hz,当指针旋转的位置在 63.2 处时,就代



表此时计算结果应该小于或等于 63.2,当指针旋转的位置位于 100 处时,就表示此时的计算结果应该大于或等于 100。指针默认指向中间点 82.4 的位置,这个位置指针的度数为 0,假设计算结果为 frequency,旋转的度数为 Degree,则两个变量之间的函数关系即旋转函数为<sup>[14]</sup>:

当  $100 \geq \text{frequency} > 82.4$  时,有:

$$\text{Degree} = 90 * ((\text{frequency} - 82.4) / (100 - 82.4))$$

当  $82.4 \geq \text{frequency} \geq 63.2$  时,有:

$\text{Degree} = -90 * ((82.4 - \text{frequency}) / (82.4 - 63.2))$



图 5 实际波形图

## 2 系统测试

在实验室环境下对系统进行测试,采用 44 100 Hz 的采样频率,直接观测 Android 应用界面显示的结果数据,并参照相关 Log 日志数据打印输出,记录下每一次调音结果中最频繁出现的数据作为实验数据。

选取十把不同的吉他分别进行相同的实验来验证所建立系统的有效性,十把吉他的统计结果见图 6。

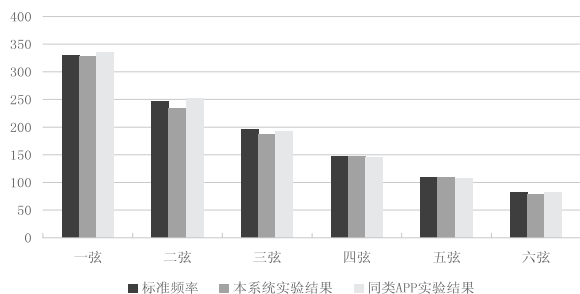


图 6 实验结果对比图

通过图 6 可以看出,所建立系统测试结果的平均误差是 2.68%,在十二平均律所允许的误差范围之内。并且与 Android 应用商店下载的同类 APP 相比而言,所建立系统总体误差相对较小,能够较好地满足用

户的使用需求。其中,所建立系统测得的一弦、四弦、五弦的频率分别为 328.1 Hz、146.6 Hz、109.1 Hz,同类 APP 测得的结果分别为 336.2 Hz、145.5 Hz、108.2 Hz,比较标准频率为 329.63 Hz、146.83 Hz、110.00 Hz,所建立系统测试结果更接近标准频率值。

## 3 结束语

为了提高吉他弦音校准精度与使用方便性,设计并建立了一种基于 Android 的吉他智能校准系统,包括系统采用的关键技术以及具体的实现流程。所建立系统通过 JNI 和线程技术,在保证系统流畅度和易用性的同时,实现了端点检测和基音频率估计的相关算法。测试验证结果表明,系统高效易用、准确度较高,具有较强的实用价值。

### 参考文献:

- [1] 武惠杰,郭天兴.高通滤波器性能研究[J].电力电容器与无功补偿,2014,35(2):5-8.
- [2] Chen Jie, Fan Chenyang, Li Bin, et al. Novel algorithm of tracking band-pass digital filter with amplitude  $1/f^2$  attenuation for vortex signal extraction[J]. Flow Measurement and Instrumentation, 2015, 45: 82-92.
- [3] 汪伟,谢皓臣,梁光明,等.加窗离散傅里叶变换性能分析和比对[J].现代电子技术,2012,35(3):115-118.
- [4] 凌菁.基于 Hanning 窗插值 FFT 的频率加权频谱分析方法[J].中国仪器仪表,2014(9):28.
- [5] 韦国刚,周萍,杨青.一种简单的噪声鲁棒性语音端点检测方法[J].测控技术,2015,34(2):31-34.
- [6] 路青起,白燕燕.基于双门限两级判决的语音端点检测方法[J].电子科技,2012,25(1):13-15.
- [7] Nibouche O, Boussakta S, Darnell M, et al. Algorithms and pipeline architectures for 2-D FFT and FFT-like transforms[J]. Digital Signal Processing, 2010, 20(4): 1072-1086.
- [8] 崔琪琳,田杜养,周燕.用离散傅里叶变换解调数字调制信号[J].通信技术,2010(2):4-6.
- [9] 何金儿,张艺鸿,蔡京玫.基于 Android 系统调音器的研究和实现[J].电子世界,2012(12):20-22.
- [10] Conti M, Nguyen V T N, Crispo B. CRePE: context-related policy enforcement for Android[C]//International conference on information security. Berlin: Springer, 2010: 331-345.
- [11] Kundu T K, Paul K. Improving Android performance and energy efficiency[C]//24th international conference on VLSI design. [s. l.]: IEEE, 2011: 256-261.
- [12] 余志龙,郑名杰. Google Android SDK 开发范例大全[M]. 第 2 版. 北京:人民邮电出版社,2010.
- [13] 纪晓阳.线程在 Android 开发中的应用[J].软件,2013(8):24-26.
- [14] 李钦华,刘芳华. Android 图形图像处理技术的研究综述[J].无线互联科技,2015(9):110-111.