ROS基础 – 实训套件

实验案例 – 教学参考手册

7 语音交互

ROS基础 – 实训套件

实验案例 – 教学参考手册

6 06 Move\_base与AMCL参数设置

ROS基础 – 实训套件

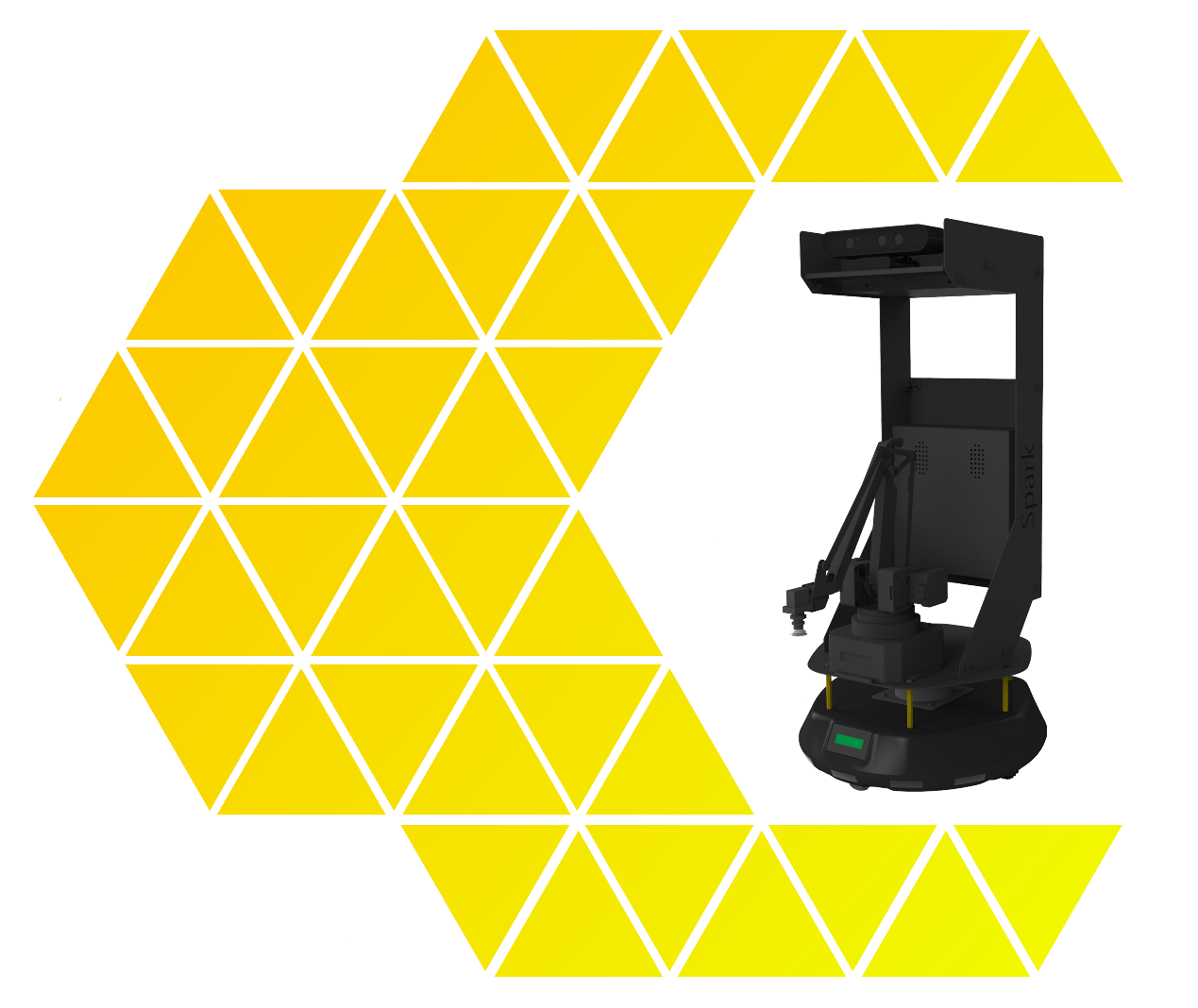
实验案例 – 教学参考手册

5 创建仿真机器人与现实机器人的同步

语音交互技术与应用

实验案例

1 语音采集与处理



**目 录**

[一、实验名称：语音采集与处理 1](#_Toc486434492)

[1、相关技能 1](#_Toc486434493)

[2、相关知识点 1](#_Toc486434494)

[3、实现效果 1](#_Toc486434495)

[4、实验要求 2](#_Toc486434496)

[5、实现思路 2](#_Toc486434497)

[6、验证与测试 4](#_Toc486434498)

[7、参考答案 4](#_Toc486434499)

# 一、实验名称：**语音采集与处理**

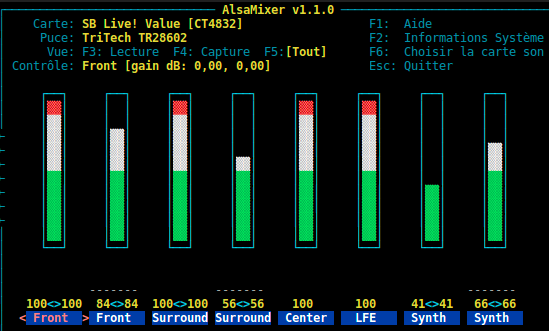
## 1、相关技能

* 可以通过接口获取音频流

## 2、相关知识点

* 音频流各项参数概念
* 音频库的使用
* 不同编解码的优缺点
* 音频处理

## 3、实现效果



1 利用AlseMixer工具调音

## 4、实验要求

**本实验要求：安装ALSA库，使用工具进行录音、播放、调音。**

* 1. 安装ALSA库 (对底层ALSA的理解参见7.0答案)
  2. 使用arecord录音工具进行录音
  3. 安装并使用sox播放工具
  4. 安装并使用alsamixer

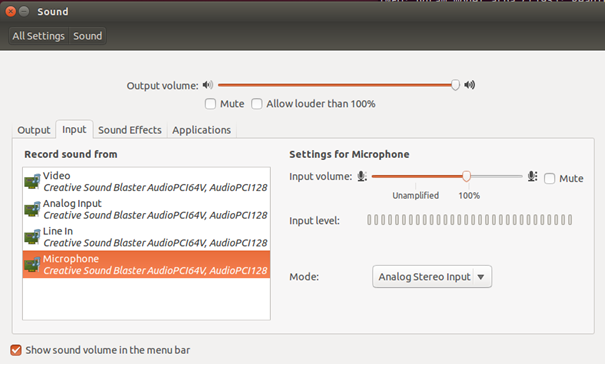
## 5、实现思路

* 1. 按实验要求，进行语音测试：
     1. 测试：

首先在系统设置里sound中input设置内置语音音量，不要太大。安装完成后我们就可以运行测试了。

首先，插入你的麦克风设备，然后在系统设置里测试麦克风是否有语音输入。

然后，运行包中的测试程序：



* 1. 按实验要求，配置安装：
     1. 系统更新

开启终端，命令行输入命令。

|  |
| --- |
| **+ 提示：**  $ sudo apt-get update  $ sudo apt-get upgrade |

* 1. 安装ALSA库

|  |
| --- |
| **+ 提示：**  $ sudo apt-get install alsa-tools alsa-oss flex zlib1g-dev libc-bin libc-dev-bin python-pexpect libasound2-dev  简要说明一下重要的库：  alsa-tools: 该库提供了对音频操作的相关指令  libasound2-dev：提供alsa应用编程API，如果使用c/c++编程会用到该库的一些函数。 |

* 1. 使用arecord录音工具进行录音

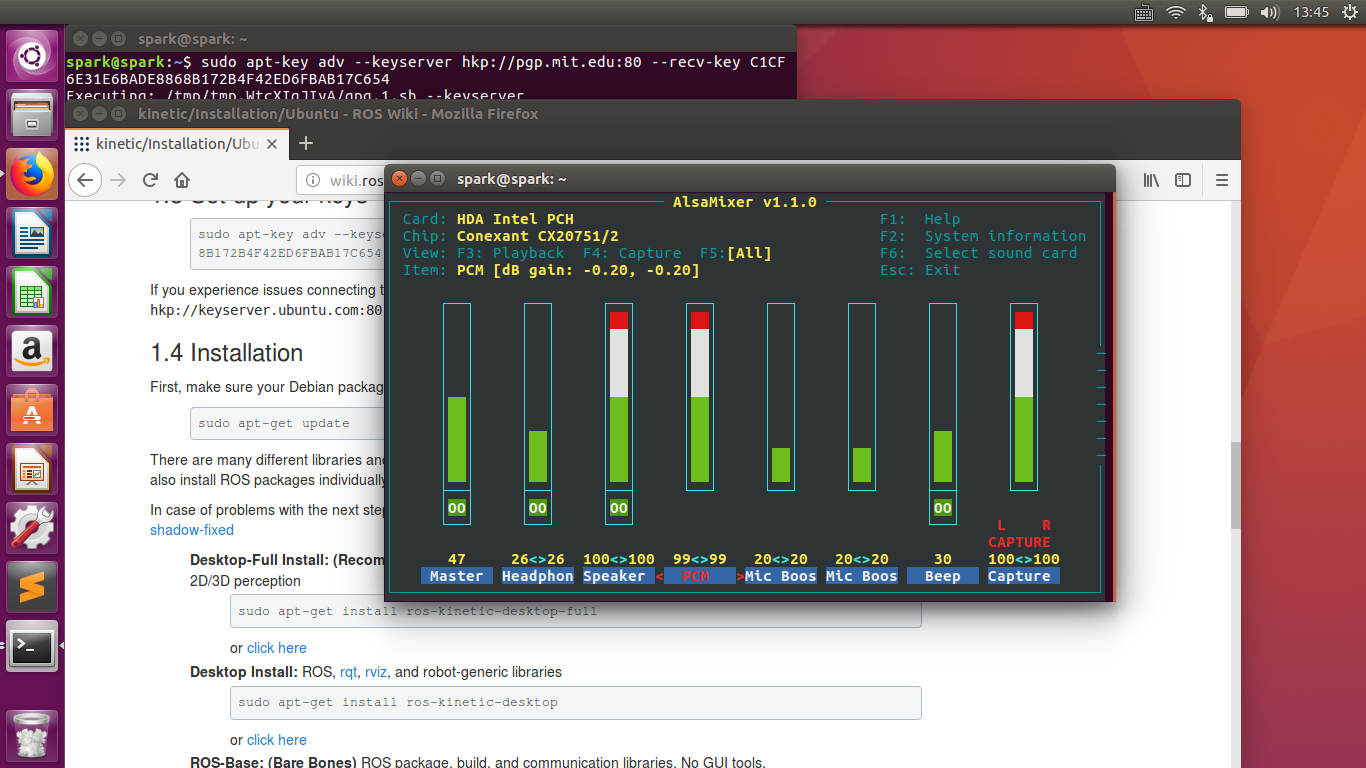
|  |
| --- |
| **+ 提示：**  $ arecord -d 10 -D plughw:0, 0 test.wav    解析：  arecord 工具为我们刚刚安装的alsa-tools提供的音频操作，即录音  -d: 录制时间（秒）  -D: 指明设备名（plughw:i,j）:其中i是卡号,j是这块声卡上的设备号 |

* 1. 安装sox播放工具

|  |
| --- |
| **+ 提示：**  $ sudo apt-get install sox |

* 1. 安装并使用alsamixer

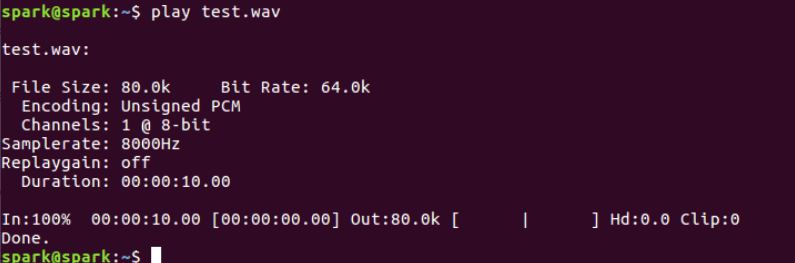
|  |
| --- |
| **+ 提示：**  $ sudo apt-get install alsamixer |



## 6、验证与测试

播放录好的音：

|  |
| --- |
| **+ 提示：**  $ play test.wav |



## 7、参考答案

一，ALSA声音编程介绍

ALSA表示高级Linux声音体系结构(Advanced Linux Sound Architecture)。它由一系列内核驱动，应用程序编译接口(API)以及支持Linux下声音的实用程序组成。这篇文章里，我将简单介绍 ALSA项目的基本框架以及它的软件组成。主要集中介绍PCM接口编程，包括您可以自动实践的程序示例。  
  
您使用ALSA的原因可能就是因为它很新，但它并不是唯一可用的声音API。如果您想完成低级的声音操作，以便能够最大化地控制声音并最大化地提高性能，或者如果您使用其它声音API没有的特性，那么ALSA是很好的选择。如果您已经写了一个音频程序，你可能想要为ALSA声卡驱动添加本地支持。如果您对音频不感兴趣，只是想播放音频文件，那么高级的API将是更好的选择，比如SDL,OpenAL以及那些桌面环境提供的工具集。另外，您只能在有ALSA 支持的Linux环境中使用ALSA。  
  
二，ALSA历史

ALSA项目发起的起因是Linux下的声卡驱动(OSS/Free drivers)没有得到积极的维护。并且落后于新的声卡技术。Jaroslav Kysela早先写了一个声卡驱动，并由此开始了ALSA项目，随便，更多的开发者加入到开发队伍中，更多的声卡得到支持，API的结构也得到了重组。  
  
Linux内核2.5在开发过程中，ALSA被合并到了官方的源码树中。在发布内核2.6后，ALSA已经内建在稳定的内核版本中并将广泛地使用。  
  
三，数字音频基础

声音由变化的气压组成。它被麦克风这样的转换器转换成电子形式。模/数(ADC)转换器将模拟电压转换成离散的样本值。声音以固定的时间间隔被采样，采样的速率称为采样率。把样本输出到数/模(DAC)转换器，比如扩音器，最后转换成原来的模拟信号。

样本大小以位来表示。样本大小是影响声音被转换成数字信号的精确程度的因素之一。另一个主要的因素是采样率。奈奎斯特(Nyquist)理论中，只要离散系统的奈奎斯特频率高于采样信号的最高频率或带宽，就可以避免混叠现象。  
  
四，ALSA基础

ALSA由许多声卡的声卡驱动程序组成，同时它也提供一个称为**libasound的API库**。应用程序开发者应该使用libasound而不是内核中的 ALSA接口。因为libasound提供最高级并且编程方便的编程接口。并且提供一个设备逻辑命名功能，这样开发者甚至不需要知道类似设备文件这样的低层接口。相反，**OSS/Free驱动是在内核系统调用级上编程**，它要求开发者提供设备文件名并且利用**ioctrl**来实现相应的功能。

为了向后兼容，ALSA提供内核模块来模拟OSS，这样之前的许多在OSS基础上开发的应用程序不需要任何改动就可以在ALSA上运行。另外，libaoss库也可以模拟OSS，而它不需要内核模块。

ALSA包含插件功能，使用插件可以扩展新的声卡驱动，包括完全用软件实现的虚拟声卡。ALSA提供一系列基于命令行的工具集，比如混音器(mixer)，音频文件播放器(aplay)，以及控制特定声卡特定属性的工具。  
  
五，ALSA体系结构

ALSA API可以分解成以下几个主要的接口：

1 控制接口：提供管理声卡注册和请求可用设备的通用功能

2 PCM接口：管理数字音频回放(playback)和录音(capture)的接口。本文后续总结重点放在这个接口上，因为它是开发数字音频程序最常用到的接口。

3 Raw MIDI接口:支持MIDI(Musical Instrument Digital Interface),标准的电子乐器。这些API提供对声卡上MIDI总线的访问。这个原始接口基于MIDI事件工作，由程序员负责管理协议以及时间处理。

4 定时器(Timer)接口：为同步音频事件提供对声卡上时间处理硬件的访问。

5 时序器(Sequencer)接口

6 混音器(Mixer)接口  
  
六，设备命名

API库使用**逻辑设备名**而不是设备文件。设备名字可以是真实的硬件名字也可以是插件名字。硬件名字使用hw:i,j这样的格式。其中i是卡号，j是这块声卡上的设备号。

第一个声音设备是hw:0,0.这个别名默认引用第一块声音设备并且在本文示例中一真会被用到。

插件使用另外的唯一名字，比如 plughw:,表示一个插件，这个插件不提供对硬件设备的访问，而是提供像采样率转换这样的软件特性，硬件本身并不支持这样的特性。  
  
七，声音缓存和数据传输

每个声卡都有一个硬件缓存区来保存记录下来的样本。当缓存区足够满时，声卡将产生一个中断。内核声卡驱动然后使用直接内存(DMA)访问通道将样本传送到内存中的应用程序缓存区。类似地，对于回放，任何应用程序使用DMA将自己的缓存区数据传送到声卡的硬件缓存区中。  
这样硬件缓存区是环缓存。也就是说当数据到达缓存区末尾时将重新回到缓存区的起始位置。ALSA维护一个指针来指向硬件缓存以及应用程序缓存区中数据操作的当前位置。从内核外部看，我们只对应用程序的缓存区感兴趣，所以本文只讨论应用程序缓存区。

应用程序缓存区的大小可以通过ALSA库函数调用来控制。**缓存区可以很大**，一次传输操作可能会导致**不可接受的延迟**，我们把它称为延时(latency)。为了解决这个问题，ALSA将缓存区拆分成一系列周期(period)(OSS/Free中叫片断fragments).ALSA以period为单元来传送数据。

一个周期(period)存储一些帧(frames)。每一帧包含时间上一个点所抓取的样本。对于立体声设备，一个帧会包含两个信道上的样本。分解过程：一个缓存区分解成周期，然后是帧，然后是样本。左右信道信息被交替地存储在一个帧内。这称为交错 (interleaved)模式。在非交错模式中，一个信道的所有样本数据存储在另外一个信道的数据之后。

八，Over and Under Run

当一个声卡活动时，数据总是连续地在**硬件缓存区**和**应用程序缓存区间**传输。但是也有例外。在录音例子中，如果应用程序读取数据不够快，循环缓存区将会被**新的数据覆盖**。这种数据的丢失被称为over run.在回放例子中，如果应用程序写入数据到缓存区中的速度不够快，缓存区将会"饿死"。这样的错误被称为"under run"。在ALSA文档中，有时将这两种情形统称为"XRUN"。适当地设计应用程序可以最小化XRUN并且可以从中恢复过来。

九，一个典型的声音程序

使用PCM的程序通常类似下面的伪代码：

打开回放或录音接口

设置硬件参数(访问模式，数据格式，信道数，采样率，等等)

while 有数据要被处理：

读PCM数据(录音)

或 写PCM数据(回放)

关闭接口