一种多媒体会议系统的实时同步混音转发算法*

李 宇[1]，郭雷勇[2]，陈建铭[2]，谭洪舟[2]
（1. 广东药学院医药信息工程学院，广东 广州 510006；
2. 中山大学信息科学与技术学院，广东 广州 510275）

摘 要：在集中式多媒体音频会议中，混音算法决定了其混音输出语音质量的高低，而混音算法的实现要依靠同步混音转发机制的支持。对 Perkins 提出的混音循环缓冲区进行分析，提出一种适用于集中式多媒体音频会议系统的实时同步混音转发算法。该算法通过设置多个循环混音缓冲区，并对其实施同步控制来实现音频数据的混音、转发功能，并利用反映操作系统调度情况的声卡缓冲区数据长度确定每次混音处理的数据长度，以减轻操作系统调度对混音的影响。实验验证了此算法的可行性和稳定性。

关键词：集中式多媒体会议系统；混音；循环缓冲区；同步；系统调度
中图分类号：TN919.8 **文献标志码**：A **文章编号**：0529-6579（2010）02-0031-06

A Real-Time Synchronous Audio Mixing and Transmitting Scheme in Multimedia Conference System

LI Yu[1], *GUO Leiyong*[2], *CHEN Jianming*[2], *TAN Hongzhou*[2]
（1. College of Medical Information Engineering,
Guangdong Pharmaceutical University, Guangzhou 510006, China;
2. School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510275, China）

Abstract: The audio mixing algorithm determines the quality of the mixing voice in the unicast based centralized multimedia conference system. However, the implementation of the mixing algorithm is based on the backup of synchronous audio mixing and transmitting mechanism. The circular mixing buffer (CMB) proposed by Perkins is analyzed, and then a real-time synchronous audio mixing and transmitting scheme is presented. This scheme is achieved by setting multi-CMBs and controlling them synchronously. The length of data used for mixing is decided by the one of data storing in the soundcard buffer which reflects operating system (OS) scheduling. This can alleviate the influence of OS scheduling on the audio mixing operation. Experiment results manifest that our scheme is feasible and steady.

Key words: centralized multimedia conference system; audio mixing; circular mixing buffer; synchronization; system scheduling

随着 VoIP 技术的发展，多媒体会议系统的应用越来越受到重视。而多媒体会议系统中的音频交互和使用要远远多于视频和数据交互，是整个多媒体会议最本质的要求。一个完整的会议系统需要考虑多种技术因素，如系统架构，网络协议，带宽消耗，计算量，混音溢出等。从会议架构和传输方式考虑，可以将多媒体会议系统分为分散式单播，分散式组播，集中式单播和集中式组播四种形式。

分散式单播对每一路终端用 unicast 传送其音频数据到会议的其他终端，编解码，混音操作在终

* 收稿日期：2009-02-01
基金项目：国家自然科学基金资助项目（60874060）
作者简介：李宇（1977年生），男，博士；通讯作者：谭洪舟；E-mail：issthz@mail.sysu.edu.cn

端完成。它要求终端具有很强处理能力的 CPU 来处理多路数据的编解码、混音等工作。这种方法要消耗大量的带宽，例如设会议有 N 个终端，则需要（$N2$-N）/2 路 unicast 连接。

根据 RFC-3550[1] 的建议，分散式组播架构的终端仅用一路 multicast 传送音频到会议其他终端，而不是每个终端都建立一个 unicast。它仅需要一路 multicast 连接，由于混音等工作仍然在终端进行，同样要求终端 CPU 具有较高的处理能力。与此同时还需要底层网络能够支持组播功能。

集中式单播架构，如 H.323 和相关文献[2-3]，通过增加一个服务器完成混音，管理等会议功能，终端源通过 unicast 发送音频给服务器，服务器对数据进行混音，再去除目标终端的音频数据，编码后用 unicast 发送给其它与会终端。它对带宽的消耗较分散式低，终端不需要进行多路的编解码和混音工作。

集中式组播架构由 Chua T. K[4] 提出，其目的是进一步减少系统对带宽的消耗。服务器对用 unicast 传送来的音频数据进行混音，然后直接用 multicast 发送到各个与会终端，若终端是活动的（即发送过自己音频数据到服务器），在发送音频数据的同时对其存储，对接收到的混音数据解码，去除自身的音频数据。若终端是非活动的，就直接解码收到的混音数据。集中式组播方法具有最少的带宽消耗，而要求终端具有存储和一定的运算能力。

在集中式单播架构的多媒体音频会议系统中混音算法决定了其输出话音质量的好坏，有多种混音算法相继被提出。常用的有平均权重算法。该算法对输入语音求算术平均来防止混音溢出，但每路音频的声量被缩小 N 倍。文献［5］提出对齐法用由各路混音音频输入在当前混音帧中最大值的绝对值 totalmax 与累加结果中最大值的绝对值 mixedmax 的比率决定的收缩因子乘以累加混音后的数据，作为一路混音输出。文献［6］提出一种自适应权重法混音算法，但因其不满足线性原理，会引入噪声，而且计算量较大。文献［7］提出非均匀波形收缩混音算法，在较好地提升音量的同时，不会产生溢出。

以上文献中的混音算法的实现都要依靠同步混音转发机制的支持，但都没有对这种同步混音转发机制进行研究和分析。针对这一问题，本文研究文献［8］中应用于分散式会议系统 Robust Audio Tools（RAT）的混音机制[9]，提出一种适用于集中式单播架构的媒体音频会议系统的实时同步混音转发算法。该算法通过设置多个循环混音缓冲区，并对其实施同步控制来实现数据的混音、转发功能，利用操作系统内部调度对混音数据的区域长度进行控制，以减轻其对混音的影响。

1 混音基本原理

对于集中式的多媒体会议系统，设会议有 N 路音频流，根据活动语音检测或说话人选择策略，将会议与会终端分成两个集合[10]。一个称为活动源集合，记为 $S_1 = \{x_i \mid i = 1,\cdots,M\}$，$M$ 表示参与混音的音频输入路数。另一个称为非活动源集合，记为 $S_2 = \{x_i \mid i = 1,\cdots,N-M\}$。设 $y_j(n)$，$x_i(n)$ 分别表示第 n 时刻的混音输出，第 i 路输入至第 j 路输出的权重，第 i 路音频流输入，则一路混音输出可表示为如下：

$$y_j(n) = \begin{cases} \mathrm{SUM} - x_j(n) & \text{if} \quad x_j(n) \in S_1 \\ \mathrm{SUM} & \text{if} \quad x_j(n) \in S_2 \end{cases} \tag{1}$$

其中 $\mathrm{SUM} = \sum_{i=1}^{M} x_i(n)$，$y_j(n)$，$x_i(n)$ 分别表示第 n 时刻混音输出采样值和音频流输入采样值。

若 $y_j(n)$ 的值超出 $[-2^{P-1}, 2^{P-1}-1]$（其中 P 为二进制位数），则产生溢出噪声。现有的各种混音算法都是以解决混音溢出问题为目的来对输入音频流进行处理的。不失一般性的做法是对音频流输入值进行加权求和或者求和后乘以一个比例因子使得混音输出值在范围 $[-2^{P-1}, 2^{P-1}-1]$ 中。目前主要有平均权重，对齐，自适应权重，非均匀波形收缩四种混音算法，其中自适应权重法由于不满足线性关系，本文不再阐述。

1.1 平均权重法

平均权重法是对所有输入采样值求和再求平均作为输出，设权重为

$$w_j(n) = \begin{cases} 1/(M-1) & \text{if} \quad x_j(n) \in S_1 \\ 1/M & \text{if} \quad x_j(n) \in S_2 \end{cases} \tag{2}$$

相应改动式（1）有

$$y_j(n)' = w_j(n) * y_j(n) = \begin{cases} (1/(M-1)) * (\mathrm{SUM} - x_j(n)) & \text{if} \quad x_j(n) \in S_1 \\ (1/M) * \mathrm{SUM} & \text{if} \quad x_j(n) \in S_2 \end{cases} \tag{3}$$

其运算量相对较少，不会产生溢出，其权重随 M 的变化会造成混音输出产生波动。

1.2 对齐法

该算法的收缩因子由各路混音音频输入在当前混音帧中最大值的绝对值 totalmax 与累加结果中最大值的绝对值 mixedmax 的比率决定。权重为

$$w_j(t) = \mu * \text{totalmax}_j/\text{mixedmax} \quad (4)$$

一路混音输出为

$$y_j(n)' = w_j(n) * y_j(n) = \begin{cases} (\mu * \text{totalmax}_j/\text{mixedmax}) * \\ (\text{SUM} - x_j(n)) & \text{if} \quad x_j(n) \in S_1 \\ (\mu * \text{totalmax}_j/\text{mixedmax}) * \\ \text{SUM} & \text{if} \quad x_j(n) \in S_2 \end{cases} \quad (5)$$

其中

$$\text{totalmax}_j = \begin{cases} \max\limits_{i,j=1,2,\cdots M \quad i\neq j}(|x_i(n)|) & \text{if} \quad x_j(n) \in S_1 \\ \max\limits_{i,j=1,2,\cdots M}(|x_i(n)|) & \text{if} \quad x_j(n) \in S_2 \end{cases}$$

$\mu \in [1, \text{totalmax}_j/\text{mixedmax}]$。

若用足够大的位数来存储 mixedmax，该方法可以避免混音溢出问题。因为溢出问题是由于混音数据播放时，声卡可处理的位数小于音频数据的实际位数引起的，而 mixedmax 只是算法处理中的某个变量，而不是最后的混音输出。

1.3　非均匀波形收缩法

文献［7］提出的非均匀波形收缩法的思想是借鉴 G.711 规范中采用的分段量化规则。算法采用分段收缩规则，低强度信号采用较大的权重，在确保信号可识别性的同时获得一定的收缩比例，而高强度信号时采用较小的权重以确保得到相应的收缩比例，该算法不产生溢出。算法的实现通过移位操作实现，不需要乘除运算，其计算量与平均权重法属于同一级别。一路混音输出为：

$$y_j(n)' = \text{sgn}(y_j(n)) \sum_{i=0}^{n_j-1} \frac{k-1}{k}\left(\frac{1}{k}\right)^i * 2^{P-1} + \frac{k-1}{k}\left(\frac{1}{k}\right)^{n_j}(y_j(n) \bmod 2^{P-1}) \quad (6)$$

$$n_j = \frac{|y_j(n)|}{2^{P-1}}$$

2　基于循环缓冲区的混音器结构

语音混合是按照某种混音算法将多路流媒体组合成一路流媒体的过程，这一过程需要开辟一个缓冲区来实现。Perkins 提出利用循环缓冲区（Circular Buffer）实现混音处理和存放混音数据[8]，并被用于分散式实时语音通信软件 RAT 中[9]。

整个循环缓冲区用累加来实现混音数据的存储。把要参加混音的音频流数据（以采样点表示）依次累加到相应的区域实现混音。缓冲区的读写由 head、tail 标志位来控制。如图 1 所示。它遵从以下规定：

（1）混音数据被写入到缓冲区，存放在 head 和 tail 所指的内存区域中；

（2）当 head 和 tail 重合时（即 head = tail），表示缓冲区中可用数据为零；

（3）当有混音数据写入时，若其数据长度（data_length）大于 head 与 tail 之间的长度（dist），则 head 向右移动至两者之差值；否则，head 不需要移动。当存储的音频流数据被读出后，tail 向右移动被读取数据长度；否则，tail 不需要移动。

当标志位到达缓冲区的最右边时发生翻转，重复利用被读取数据的地址段实现缓冲区的循环使用。这使得被读出的数据分成两段，一段位于循环缓冲区终点位置，另一段位于开始位置，失去了连续性。一种保持读取数据连续性的做法是在原有缓冲区的两边分别增加存储空间，当读到翻转点时把另一段的数据复制到翻转点后的区域，保持数据存放的连续性，如图 2 所示。

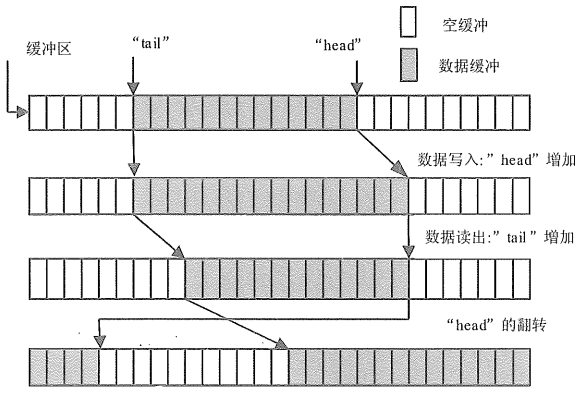

图 1　循环缓冲区的实现

Fig. 1　Implementation of a circular mix buffer

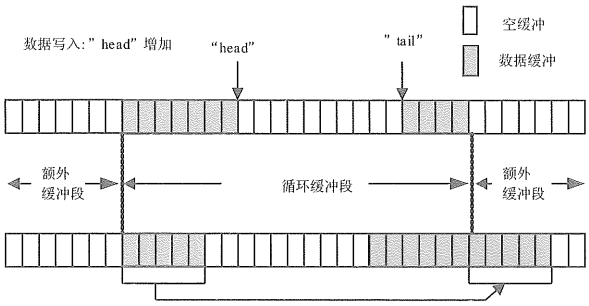

图 2　带额外存储区域的循环缓冲区的实现

Fig. 2　Implementation of a circular mix buffer with additional copy buffer

我们用设计含有循环缓冲区的混音器来实现数据的混合，写入和读取操作。一个含有循环缓冲区的混音器可由以下数据结构表示：

```
struct s_mixer{
    int             buf_len;
    int             head, tail;
    timestamp_t     head_time, tail_time;
    timestamp_t     this_read_avail_time;
    int             dist;
    sample          * mix_buffer;
    uint32_t        sample_rate;
    uint16_t        channels;
    uint32_t        buffer_length;
```

其中 buf_len 为缓冲区的长度，head 和 tail 分别是缓冲区头部和尾部的样本索引，与时间戳 head_time 和 tail_time 存在对应关系。this_read_avial_time 表示本次读取操作可读取到的数据末端的时间戳，dist 为 head 与 tail 之间的距离，mix_buffer 指缓冲区，sample_rate，channels 和 buffer_length 分别是采样率，声道数，缓冲区长度。

3 同步混音转发算法

对于分散式多媒体会议系统，混音处理在终端进行，处理后的混音数据在终端播出，不再发送，而对于集中式单播架构多媒体会议系统，混音在服务器完成，需要同步转发。这使得同步混音转发功能的实现需要考虑以下三个问题：

（1）与会终端在 S1 和 S2 集合中来回切换对混音操作的影响；

（2）各个混音器之间的同步问题；

（3）操作系统调度对混音的影响。

针对上述问题，通过设置多个具有前面所述数据结构的混音器，并对其实施同步控制，依据系统调度决定每次混音处理的数据长度，消除以上因素对混音的影响。

3.1 算法描述

设与会终端个数为 N，某一时刻 S_1 的个数为 M，则 S_2 的个数为 N-M。设置一个公共混音器（ms），对每个与会终端分别分配一个混音器（ms_src）。根据减法混音方式，公共混音器 ms 负责所有活动源的混音数据的存储、控制；而 ms_src 则负责转发回自身的混音数据的存储、控制。算法用到的时间统一用 RTP 协议定义的时间戳[1]。设 cur_ts 为当前系统时间戳，audio_device_read（）为读取声卡缓冲区数据函数。

算法的开始与结束由 Should_exit 标志位控制。从而使上级程序能够利用它实现对算法的开始与结束的控制。程序先从声卡或虚拟声卡获得本次混音转发能够处理的数据长度。推进公共混音器的头部时间，使得头部与尾部之间的时间距离为混音转发数据的时间长度。混音转发算法流程伪代码如下：

```
设置各个混音器的 tail_time, head_time 为当前系统时间 cur_ts
While ! Should_exit {
  调用 elapsed_time = audio_device_read( )获得本次混音的时间
    长度;
  从端口获得 RTP/RTCP 数据包;
  ms->head_time ⇓ ms->head_time + elapsed_time;
  for active_source(i) {
    for each packet {
      计算每帧语音的播出时间 playout;
      依照 speech samples 的 playout 和长度 datalength,
        计算播放结束时间 playout_end;
      对语音解码获得采样值 speech samples;
      累加 speech samples 到 ms->buffer;
      if (playout_end > ms->head_time)
        ms->head_time ⇓ playout_end,推进 ms->head;
      计算写入区间长度,写入属于该活动源的 speech samples
      到 ms_src(i)->buffer;
      if (playout_end > ms_src(i) > head_time)
        ms_src(i)->head_time ⇓ playout_end,
          推进 ms_src(i)->head;
    }
  }
  依据 elasped_time,ms->tail, 从 ms->buffer 中读取公共混音
    数据段 cdata;
  for active_source(i) {
    对存放 ms_src(i)->buffer 的区域[ms_src(i)->tail,
      ms_src(i)->tail + length(cdata)]中的数据取反,与 cdata 相
加,
      实现减法运算;
  }
  for inactive_source(i) {
    对存放 ms_src(i)->buffer 的区域[ms_src(i)->tail,
      ms_src(i)->tail + length(cdata)]写入 commixdata;
    if (playout_end > ms_src(i)->head_time)
      ms_src(i)->head_time ⇓ playout_end,推进 ms_src(i)->head;
  }
  for active_source(i)  {
    从 ms_src(i)->buffer 读取 length(cdata)长度的数据,
      发送数据;
    推进 ms_src(i)->tail;
  }
  for inactive_source(i)  {
    从 ms_src(i)->buffer 读取 length(cdata)长度的数据,
      发送数据;
    推进 ms_src(i)->tail ;
  }
}。
```

3.1.1 混音阶段 从网络端口接收来自活动源的 RTP/RTCP 数据包。对于每一个活动源，计算每一帧的播出时间戳 playout_time 和播放结束时间戳

playout_end，信道解码（若有信道编码），信源解码，得到音频波形数据，计算写入区间长度，分别把波形数据累加到缓冲区和写入自身混音器缓冲区中的相应区间。推进各自的标志位 head（head 和 head_time 具有对应关系[7]）。

依据时间戳 elaspe_time 换算出数据长度，从公共混音器缓冲区读出混音数据 cdata，利用读出的混音数据减去各个活动源混音器缓冲区［tail，tail + datalength］区域的数据。而对于非活动源，直接把 cdata 复制到其混音器缓冲区［tail，tail + datalength］区域，推进其 head。这时，公共混音器和非活动源混音器存储着所有活动源的混音数据，而活动源混音器存储着不含自身语音数据的混音数据。

3.1.2　*发送阶段*　对于活动源或非活动源，从各自混音器缓冲区读出与长度为 datalength 的数据，送往网络端口，推进标志位 tail。

3.2　算法说明

3.2.1　*非活动源标志位的更新*　当与会终端属于活动源时，由于有自身的语音数据写入和读出，其标志位 head，tail 被更新；当与会终端属于非活动源时，由公共混音器复制混音数据写入和读出，因而使其 head，tail 更新。这使得非活动源终端不会因为自身没有数据参与混音处理，导致其 head，tail 得不到更新，导致当其成为活动源时没有正确的 head_time 和 tail_time 可用。

3.2.2　*混音器缓冲区写入区间的确定*　记前次混音的时间戳 playout_end 的值为 nextmix。先比较 nextmix 与当前 playout 的大小，用二者的最大值作为起点时间戳，当前 playout_end 做为结束时间戳，换算出写入区间的起点和终点，写入数据。当 nextmix 大于 playout，表示有部分数据提前到达，这部分数据不能写入缓冲区，应该丢弃。Playout 的计算可参考文献［11］。

3.2.3　*每次混音的数据长度*　每次需要混音处理的数据长度可以通过操作系统调度程度决定，声卡设备中累积的数据量间接反映了操作系统的调度情况。正常调度情况下，每次读取到的数据长度应该是一帧时间的数据量。当声卡累积的数据量大时，反映出操作系统调度任务量大，因而读取到的数据量能够反映出系统调度情况。通过对声卡（或虚拟声卡）读取数据，计算出需要处理数据的等效时间。这使得在每次混音转发处理过程中，各个混音器中被处理的数据长度是相等的，保证了各个 tail_time 在每次混音数据被写入前和被读出后都是相等的。因此混音、转发的同步控制得以实现。

4　实验与分析

实验用配置为 CPU P4 2.4 G、内存 1 G 的服务器和配置为 CPU P4 2.4 G、内存 512 M 的终端组建了一个具有 9 个与会终端的集中式单播会议系统。考虑到一般会议中很少会出现超过 3 个人同时说话的情况这一事实，实验中会议系统只用到 9 个与会终端。测试几十个与会终端同时说话混音的情况没有太大的实用意义。服务器与终端之间用百兆网络连接，音频流格式统一为 G.711Ulaw，8 kHz 采样，8 位，单声道。本算法不是研究混音溢出问题，因此没有必要进行混音前和混音后的音频波形图比较。会议采用了平均权重法防止溢出处理，进行了会议系统的 CPU 负载和内存的使用两项测试来检验会议系统的稳定性。

图 3 表示 CPU 一直运行平稳，没有出现较大范围波动。图 4 表示随着与会终端成为活动源的个数的增加，内存的使用情况。内存的使用量与活动源数量成线性递增关系。实验中测得当加入一个活动源终端后，内存增加 240 kB；当一个活动源变成非活动源后，内存减少 40 kB。

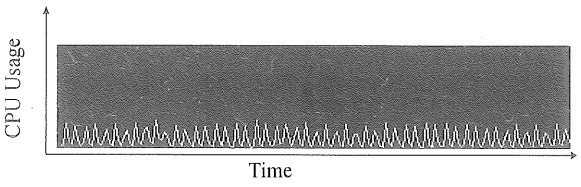

图 3　CPU 使用率

Fig. 3　CPU Usage

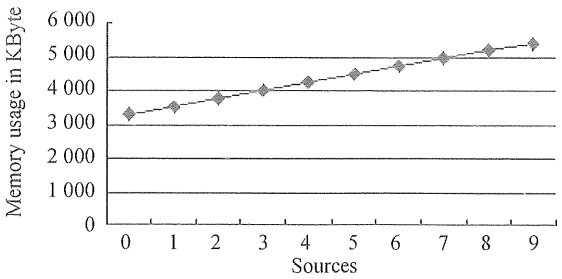

图 4　内存占用情况

Fig. 4　Memory lsage

5　结　论

本文提出了一种适用于集中式单播架构的多媒体音频会议系统的实时同步混音转发算法。该算法

可实现混音输出数据的同步转发功能，能够在一定程度上减少操作系统内部调度对混音的影响。而且该算法可串接各种混音算法如对齐法，非均匀波形收缩法等，解决混音溢出问题，结合说话人选择策略如文献［12］，可应用于大型会议系统实现混音转发功能。实验结果验证了该算法的稳定性和有效性，表明该方法是可行的。

参考文献：

［1］ SCHULZRINNE H, CANER S, FREDERICK R, et al. RFC 3550: RTP: A transport protocol for real-time applications[S]. IETF, 2003.

［2］ ITU-T Rec. H. 323, Packet-based multimedia communication system, v4[S]. ITU, 2000.

［3］ HAWWA S. Audio mixing for centralized conferences in a SIP environment［C］// Multimedia and Expo2002. Proceedings, Lausanne, Switzerland, 2002, 8: 269 - 272.

［4］ CHUA T K, PHEANIS D C. Bandwidth-conserving real-time VoIP teleconference system［C］// Proceedings of the Third International Conference on Information Technology: New Generations, Las Vegas, USA, 2006, 4: 535 - 540.

［5］ 杨树堂，余胜生，周敬利. 基于分组网络的多点实时语音及调度算法［J］. 软件学报，2001，12(9)：413 - 1419.

［6］ 樊星，顾伟康，叶秀清. 多媒体会议中的快速实时自适应混音方案研究［J］. 软件学报，2005，16(1)：108 - 115.

［7］ 王文林，廖建新，朱晓民. 多媒体会议中新型快速实时混音算法[J]. 电子与信息学报，2007，29(13)：690 - 695

［8］ PERKINS C. RTP: Audio and video for the internet［M］. Boston: Addison Wesley, 2003.

［9］ PERKINS C, HODSON O. Robust audio tools［CP/OL］.［2004 - 01 - 01］. http://www-mice. cs. ucl. ac. uk/multimedia/software/rat/index. htm.

［10］ BENYASSINE A, SHLOMOT E, SU H. ITU-T recommendation G. 729, annex B, a silence compression scheme for use with G. 729 optimized for V. 70 digital simultaneous voice and data applications［J］. IEEE Communications Magazine, 1997, 35(9)：64 - 73.

［11］ RAMJEE R, KUROSE J, TOWSLEY D, et al. Adaptive playout mechanisms for packetized audio applications in wide-area networks［C］// Proceedings of the Conference on Computer Communications, Toronto, Canada, 1994, 6 : 680 - 688.

［12］ 涂卫平，胡瑞敏，艾浩军. 视频会议中音频多点处理研究［J］. 武汉大学学报：信息科学，2002，27(1)：98 - 101.

（上接第30页）

［4］ LI X C, OLAFSDOTTIR G. Development and application of electronic noses in freshness assessment of fishery products［J］. Journal of Fisheries of China, 2002, 26(3): 275 - 280.

［5］ LUO D H, HOSSEINI H G, STEWART J R. Application of ANN with extracted parameters from an electronic nose in cigarette brand identification［J］. Sensors and Actuators B, 2004, 99:253 - 257.

［6］ PETERSSON H. Signal processing for chemical multi sensor systems［C］. Project Preparatory Paper, 2004.

［7］ SRIVASTAVA A K. Detection of volatile organic compounds (VOCs) using SnO2 gas-sensor array and artificial neural network［J］. Sens. Actuators B: Chem, 2003, 96(1/2):24 - 37.

［8］ ZHANG H X, BALABAN M O, PRINCIPE J C. Improving pattern recognition of electronic nose data with time-delay neural networks［J］. Sens Actuators B: Chem, 2003, 96(1/2):385 - 389.

［9］ 潘胤飞. 基于非线性模式识别的电子鼻技术在苹果分类中的应用[D]. 镇江：江苏大学, 2003.

［10］ LINDER R, PÖPPL S J. A new neural network approach classifies olfactory signals with high accuracy［J］. Food Quality and Preference, 2003, 14(5/6):435 - 440.

［11］ LIAO S H, WEN C H. Artificial neural networks classification and clustering of methodologies and applications-literature analysis from 1995 to 2005［J］. Expert Systems with Applications, 2007, 32(1): 1 - 11.

［12］ GUTIERREZ-OSUNA R. Member, IEEE, pattern analysis for machine olfaction: a review［J］. IEEE Sensors Journal, 2002, 2(3):189 - 200.