

中科院声学所在线识别引擎

数据结构设计

目录

[1.概述 1](#_Toc89087839)

[1.1简介 1](#_Toc89087840)

[1.2 读者对象 1](#_Toc89087841)

[1.3术语定义 1](#_Toc89087842)

[1.4 参考资料 1](#_Toc89087843)

[2. 数据结构设计 1](#_Toc89087844)

[2.1 语音数据信息结构体 1](#_Toc89087845)

[2.1.1 语音数据格式 1](#_Toc89087846)

[2.2 识别结果输出数据结构 1](#_Toc89087847)

[2.2.1 分段任务/结果信息数据结构 1](#_Toc89087848)

[2.2.2 识别结果信息数据结构 1](#_Toc89087849)

[2.3 识别引擎状态数据结构 1](#_Toc89087850)

[2.4 识别热词功能数据结果 1](#_Toc89087851)

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| 1.0 | 陈向东 | 龚云波  黎塔 | 2021/04/28 |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 1.概述

1

1.1简介

“好记性不如烂笔头”，反映出人们对记录的重视程度。从传统的  
纸文化进步到计算机时代，除了手写识别，当信息量更大的时候，人们  
会选择音频和视频。虽然人们不用担心存储容量不够用，但是今后如何  
有效的利用这些多媒体的记录，成为一个突出问题。

很多人为此倾向花高价邀请速录员。一来需要成本，二来在专业领  
域，培训和保密又成为新的问题。现在，用户有了新的选择-声学所识别引擎。该识别引擎不仅能保存语音，还能把语音转化成文字与之对应保存下来。有了它，今后无论何时用户想要调出所需的记录，只要通过查找相应文字并简单点击，相关的语音片断就找到了。同时，本引擎还提供了直接识别以往录音的功能，把以前宝贵的资料统统转化成文本，给用户的媒体档案库将来查询时提供最大便利。

中科院声学所连续语音识别引擎，针对连续音频流（即来自说话人直接录入的语音，或者广播电台、电视台或其他领域的音频信号）进行识别，将音频信息自动转化成文字。覆盖汉语中绝大多数词语，适用于说普通话的任何人群。输出的结果都是汉字，兼容数字。

在输入的声音中，检测出可靠的语音，排除噪声音乐等，实时送入语音识别解码器进行识别。识别引擎把音视频中提取出的语音分成 25 毫秒一帧，提取有用特征，然后识别出一些类似拼音的结果（声学模型），再根据汉语字词句之间的搭配概率（语言模型），综合考虑。当然，考虑的越多（beam路径越大），识别准确率相对提高，同时消耗的时间就增长了；所以我们有优化策略，及时排除不可能的结果，避免系统过慢。用户可以通过调节这些参数来平衡识别质量和速度，以满足实际的需要。

最后返回的识别结果，软件以汉语中基本词语为单位给出了包含对应时间点的词序列。

需要说明的是，对识别正确率以及识别速度来说，输入语音（普通话）的质量（录音过程）是很重要的。录音时应尽可能的排除噪声和音乐、增大语音，识别效果就会好得多，因此建议用户在录音的时候选用质量较好的麦克风。

另外，系统在检测是否有语音信号时可能会带来一定的时间滞后，用户在实时录入时在每句话结束后，需要略有停顿，这样可以提高系统语音检测的速度和精度。语音检测的越好。识别引擎的识别效果也就越好。

## 1.2 读者对象

本文档是对识别引擎的内部代码流程和相关技术点的整体说明，因此读者最好满足如下几点条件后再读此文档：

* 有一定的C和C++语言基础
* 了解语音识别过程和原理
* 了解声学所识别引擎实现原理

## 1.3术语定义

TBNR，Thinkit-Broadcast-News-Recongnize：中科信利识别引擎

CN，Confuse-Network：混淆网络

VAD，Speech Activity Detection：语音活动检测

WFST, Weighted Finite-State Transducer：加权有限状态转录即

ITN,Inverse –Text-Normalization：逆文本标准化

# 2. 数据结构设计

2

声学所识别引擎主要数据结构设计如下：

## 2.1 语音数据信息结构体

## 2.1.1 语音数据格式

enum SPEECHTYPE

{

TSR\_ALAW\_PCM = 0, //alaw

TSR\_ULAW\_PCM = 1,//ulaw

TSR\_LINEAR\_PCM = 2,//linear pcm

TSR\_ADPCM = 3,//AD pcm

TSR\_RAW\_16 = 4 | TSR\_LINEAR\_PCM, //8k\_16bit\_pcm

TSR\_16K\_16 = 8 | TSR\_RAW\_16, //16k\_16bit\_pcm

TSR\_MFCC = 16, //MFCC feature

TSR\_MFCCPLP = 17,//以下数据为预留格式

TSR\_RASTAPLP = 18,

TSR\_WITHEAD = 19,

TSR\_MFCC\_FE = 8 | TSR\_MFCC,

TSR\_MFCCPLP\_FE = 8 | TSR\_MFCCPLP,

TSR\_RASTAPLP\_FE = 8 | TSR\_RASTAPLP,

TSR\_WITHEAD\_FE = 8 | TSR\_WITHEAD,

TSR\_SILENCE = 32,

TSR\_LOST = 33,

};

## 2.2 识别结果输出数据结构

## 2.2.1 分段任务/结果信息数据结构

struct Task

{

void \*newSet; //作为任务时表示特征数据，作为结果时表示识别结果WordResult

void \*latSet; //作为任务时表示二次判决开启情况下该分段是否为有效特征数据

int frameNum; //作为任务时表示特征数据大小

long long sessionId; //表示该任务/结果对应的会话ID

unsigned long saveFileIdx; //表示该任务/结果对应的分段ID

unsigned long dataTimestamp; //表示该任务/结果对应的分段的起始位置

long decoderId; //作为任务时表示该任务在该段的位置，作为结果时表示对应的解码ID

unsigned long dataTimestampEnd; //表示该任务/结果对应的分段的结束位置

SOUND\_TYPE soundType;// 作为结果时该结果在分段中的位置

};

enum SOUND\_TYPE

{

MALE=0, //男，在结果中表示该将结果为分段中间识别结果

FEMALE, //女，在结果中表示该将结果为分段最终识别结果

MUSIC, //音乐

NOISE, //噪声

UNK,

};

## 2.2.2 识别结果信息数据结构

识别结果数据结构

struct WordResult

{

int candNum;//该词可选个数+1，至少为1不固定，首个为最终结果1-best，通常为第一选

ACand \*cands;//识别结果结构体

WordResult \*next;

WordResult()

{

candNum = 0;

cands = 0;

next = 0;

}

};

候选识别结果数据结构

struct ACand

{

char \*text; //候选文本结果

float score; //候选得分，0.00~100.00，candNum个

float startTime; //开始时刻，单位为秒s

float endTime; //结束时刻，单位为秒s

ACand \*next; //到下一个的指针

#ifdef ENHANCED\_RESULT

char \* phone; //候选分词形式结果

char \* segTime; //候选分词时间结果

char \* confidence; //候选结果得分

#endif

ACand ()

{

text = 0;

next = 0;

score = startTime = endTime = 0;

phone = 0;

segTime = 0;

confidence = 0;

};

};

## 2.3 识别引擎状态数据结构

引擎事件信息，即时反馈引擎状态

enum TBNR\_EVENT

{

TBNR\_EVENT\_START\_COMPLETE = 0, // 启动识别完成

TBNR\_EVENT\_STOP\_COMPLETE, // 停止识别完成

TBNR\_EVENT\_RECOGNITION\_COMPLETE, // 识别完成

TBNR\_EVENT\_RECOGNITION\_DOING, // 识别中

TBNR\_EVENT\_RECOGNITION\_IDLE, // 空闲

TBNR\_EVENT\_RECOGNITION\_ERROR, // 异常

};

## 2.4 识别热词功能数据结果

指定输入热词的发音方式

typedef enum HOTWORD\_PRON\_MODE

{

AUTO\_GEN\_PRON\_MODE = 0, //自动获取热词发音

USER\_GEN\_PRON\_MODE//用户自定义热词发音

}HOTWORD\_PRON\_MODE;

//热词信息

typedef struct HOTWORD

{

char szHotWordChinese[256]; //热词

char szHotWordPinyin[256]; //热词发音，自动获取发音时该字段可以为空

int pronmode; //热词发音方式

float threshold; //热词权重，权重越大用容易识别出热词

#ifdef \_\_cplusplus

HOTWORD()

{

pronmode = AUTO\_GEN\_PRON\_MODE;

szHotWordChinese[0]='\0';

szHotWordPinyin[0]='\0';

threshold=0.0;

}

#endif

}HotWord;