嵌入式连续语音识别系统

设计说明书

1. 引言

编写详细设计说明书是软件开发过程必不可少的部分，其目的是为了使开发人员在完成概要设计说明书的基础上完成概要设计规定的各项模块的具体实现的设计工作。

1. 软件整体设计
	1. 软件需求概括

本软件采用传统的软件开发生命周期的方法，采用自顶向下，逐步求精的结构化的软件设计方法。

本软件主要有以下几方面的功能

* 语音接收。
* 特征提取。
* 模型计算。
* 解码。
* 结果输出。

定义

* + 1. 连续语音识别功能。语音识别，即：输入语音信号（.wav格式音频数据流，可由录音得到，也可读取文件得到），输出文本（字符串格式）。文本的内容，就是音频里面说的内容（类似微信语音输入）。语音识别可分为孤立词识别和连续语音识别。孤立词识别只能识别事先预定好的命令词中的一个（一般几十个或一百个词），连续语音识别则可以识别任意词组成的，任意长度的句子。
		2. 嵌入式语音识别。语音识别系统从实现方式上，可以分为两大类：

一是在线语音识别（也叫云端语音识别）。这种方式，终端负责录音，之后把数据上传至远程服务器，服务器进行语音识别的计算，最后把识别结果发送回终端，终端再显示出来。目前大部分手机上的语音助手都是这种方式，断网就无法使用。这么做的原因是，语音识别需要较强的计算能力，和较大的内存，一般终端计算能力有限。嵌入式语音识别，是指能在运算能力较弱的嵌入式设备运行的语音识别系统，它是一个本地端的语音识别系统，完全是在本地运行、计算的，无需数据上传和服务器。

* 1. 需求概述：
1. 需要进行连续语音识别。
2. 可在离线状态下完成语音识别。
3. 可以通过不同平台编译运行在过平台上。
4. 结构清晰，运行效率高。
	1. 限制条件：

系统开发的条件是普通PC以及相对应的系统，本次开发所用的系统是linux。

* 1. 软件总体结构：

 嵌入式连续语音识别系统主要分为5各模块如下图1所示：



图1:嵌入式连续语音系统结构

* **语音接收：**语音接收模块负责接收外部数字化的音频文件，并对文件进行格式检测、校验等工作。嵌入式连续语音系统中完成的异步连续接收处理语音的能力主要体现在该模块。接收模块采用多线程方式将连续语音信号以固定的语音间隔特征进行缓冲、分割，但有序向后传递语音数据块。
* **特征提取：**特征提取是识别的重要模块，它将从语音接收模块得到的语音块进行预处理，并从中提取出可以进行计算的特征值。特征提取通过预加重、分帧、加窗、傅里叶变换、梅尔滤波等等过程。
* **模型计算：**模型计算是识别的核心模块，它将语音块中的特征值作为输出，通过预先训练出的声学模型计算出识别值。
* **解码：**解码模块是识别另一个核心模块，它将模型计算结果进行解码，得到对应的识别信息。
* **结果输出：**输出模块是嵌入式连续语音识别的最后一个模块，它将识别后的信息进行格式化封装，以UTF-8格式输出。
1. 软件功能描述
2. 语音接收

 

图2：语音接收流程图

语音接收部分是通过异步模式进行实时语音文件接收，如图2所示流程。

1. 数字语音文件流接收：
2. 格式识别:对流入的数字音频文件进行格式校验，非法格式拒绝写入缓冲池。
3. 缓冲池:是一块多线程共享空间，语音文件写入时，监控机同步读取内容。
4. 监控机:负责监控缓冲池，当缓冲池不为空时，寻求截取点，当需求到截取点后，分线程完成截取、校验、输出、是否缓冲池。监控机记录线程顺序，保证有序输出。
5. 截取：子线程内部工作，在制定标记空间内获取一段语音文件。
6. 文件校验：子线程内部工作，对截取后的语音文件进行完整性校验，如果不合法，将进行重新截取。
7. 输出：子线程内部工作，以净格式音频文件段输出给特征提取模块，输出后将释放缓冲池中的空间。
8. 特征提取



图3：特征提取流程图

1. 预加重：

void Preemphasize (VectorBase <BaseFloat> \*waveform, BaseFloat preemph\_coeff);

1. 加窗：

void ExtractWindow(int64 sample\_offset, const VectorBase<BaseFloat> &wave, int32 f, // with 0 <= f < NumFrames(feats, opts) const FrameExtractionOptions &opts, const FeatureWindowFunction &window\_function, Vector<BaseFloat> \*window, BaseFloat \*log\_energy\_pre\_window);

1. 傅里叶变换、计算频谱能量：

void ComputePowerSpectrum(VectorBase<BaseFloat> \*complex\_fft);

1. 梅尔滤波：

void Compute(const VectorBase<BaseFloat> &fft\_energies, VectorBase<BaseFloat> \*mel\_energies\_out) const;

1. 模型计算



图4：模型计算流程图

1. 矩阵乘法：

void AddMatMat(const Real alpha,const MatrixBase<Real>&A, MatrixTransposeType transA,const MatrixBase<Real>&B, MatrixTransposeType transB, const Real beta);

1. 矩阵随机初始化：

void SetRandn();

1. 矩阵置零：

void SetZero();

1. 取矩阵某一列：

void CopyColFromVec(const VectorBase<Real> &v, const MatrixIndexT col);

1. 求矩阵中最大的数：

Real Max() const;

1. 矩阵所有元素求和：

Real Sum() const;

1. 矩阵每个元素求对数：

void ApplyLog();

1. 矩阵每个元素求指数：

void ApplyExp();

1. 向量加向量：

void Add(Real c);

1. 向量相乘：

void AddMat(const Real alpha, const MatrixBase<Real> &M,

 MatrixTransposeType transA = kNoTrans);

1. 读取模型：

void Read(const std::string &rxfilename);

1. 读取先验概率：

explicit PdfPrior(const PdfPriorOptions &opts);

1. 神经网络前向计算：

void Propagate(const MatrixBase<BaseFloat> &in,

 Matrix<BaseFloat> \*out);

1. 解码器



图5：解码器流程图

1. 读取解码图：

VectorFst<StdArc> \*ReadFst(const std::string fst\_hclg\_string);

1. 读取词表：

std::vector<std::string> ReadWords(const std::string words\_table\_string);

1. 解码器初始化：

bool Init(const char \*fst\_hclg, const char \*words\_table);

1. 解码：

void Decode(DecodableInterface \*decodable);

1. 解码结果：

bool GetResult(std::string \*result);

1. 结果输出

 

图6：结果流程图

结果输出部分是将大量异步且有序的识别结果按序输出过程，如图5所示流程。

1. 接收解码器的解码结果：
2. 结果输出集:由于从特征提取、模型计算、解码过程是异步进行的，但语音结果是有序的，所以解码结果直接输出是不可行的，所以通过结果输出集进行排序、等待。
3. 文字输出:以utf-8的格式进行输出。
4. 软件接口设计

软件对外暴露了4个接口函数：

decoder\_init() 是引擎初始化函数，需指定模型文件和一些参数；

decoder\_put\_data() 是识别函数，需向系统提供数据，让系统进行计算；

decoder\_get\_result() 是获取结果的函数，可获取识别结果；

decoder\_release() 是引擎释放函数。