Voice-Controlled Calculator

implemented by CUMSphinx and JAVA

Chenqi Zhang

Department of Electrical and Computer Engineering

FAMU-FSU College of Engineering

Florida, The United States

cz14b@my.fsu.edu

*Abstract*—This technical document is a class research project report of EEL 4930/5930 Speech-Processing . CMU Sphinx is a group of speech recognition systems developed at Carnegie Mellon University. The speech recognition engine and acoustic model trainer of Sphinx make it very easy to develop speech-based applications. In this project, first I go through the developer's tutorial provided by the official website. In particular, I learned building the application with Sphinx4, which is the toolkit for developing in JAVA. Then I write a Voice-Controlled calculator, analysis the performance and propose some ideas about further improvement.

Keywords—CMU Sphinx; JAVA; Speech Recognition; Voice-Controlled calculator

# Introduction

Speech recognition is one of the most hot topics nowadays. We can easily find many applications based on speech recognition, such as Voice-to-Text messages, voice memo, language translation system, the control of SmartTV, and so on. The common ideas of those applications is 1)get the input voice instructions; 2)try to analysis and understand those "physical sound", mapping the sounds to specific meanings; 3)execute those instructions and provide users with output. Apparently, the most difficult part is the second phase, speech recognition. There are bunches of advanced techniques and complex statistic and probability theories in the process of speech recognition. However, the CMU Sphinx handles the most difficult part for us. It is like a package with useful interfaces which can be used in the application. With the help of Sphinx4, I implement a Voice-Controlled calculator in JAVA. When the calculator is under operation, the user don't need to push the button. Instead, the user gives all the commands by voice. For example, when the user says "one two one three", "plus", "five six nine", the result will appear on the display panel as a string "1213+569=1782". From my point of view, such a calculator is sometimes useful. The most significant use is that it is suitable for people with hand disability. Another scenario is that when people are doing experiment with their hands, the Voice-Controlled calculator will give them great convenience and set free their hands.

# CMU SPHINX

This section gives an overview of the CMU Sphinx. Most of the materials are from CMU Sphinx official website: http://cmusphinx.sourceforge.net/wiki/tutorial

## Basic Concept of Speech

Speech is a complex phenomenon. A naive perception is that Speech is built with words consists of phones. However, the reality is that Speech is a dynamic process without clearly distinguished parts. All modern descriptions of speech are to some degree probabilistic. That means no certain boundaries between units, or between words. Speech to text translation and other applications of speech are never 100% correct.

## Maintaining the Integrity of the Specifications

CMU Sphinx toolkit is a leading speech recognition toolkit with various tools used to build speech applications. CMU Sphinx toolkit has a number of packages for different tasks and applications. It’s sometimes confusing what to choose. To cleanup, here is the list:

Pocketsphinx — lightweight recognizer library written in C.

Sphinxbase — support library required by Pocketsphinx.

Sphinx4 — adjustable, modifiable recognizer written in Java.

Sphinxtrain — acoustic model training tools.

# DEVELOPMENT PROCEDURE

This section presents the application developing procedures, including environment configuration, the function modules design, and coding as well as debug.

## Environment Configuration

In order to develop the application with Java and Sphinx4, there is the requirement list:

* Java SE Development Kit(JDK)8 and Java SE Runtime Environment(JRE)8
* Eclipse SDK. We can achieve great efficiency coding with Eclipse
* Sphinx4. Java speech recognition library.
* Java Speech API(JSAP). This allows Java application to incorporate speech technology into their user interfaces.

First, download the latest available released CMP Sphinx toolkit from the official website:

http://cmusphinx.sourceforge.net/wiki/download.

CMU Sphinx toolkit has a number of packets for different tasks and application. What we need is Sphinx4, an adjustable, modifiable recognizer written in Java. I will pass the process about how to configure JDK and JRE, and just focus on the part about configuration of Sphinx4.

The JSAP file is already in the download Sphinx4 packet, find it and setup it.

Now let's start with something simple to get familiar with Sphinx4. I will fulfill the 'Hello World' demo attached with the Sphinx4 packet.

First, create a new project in Java. Before programming we need to import some useful JAR(Java Archive). JAR is a packet file format typically used to aggregate many Java class files and associated data and resources. Creating a library named Sphinx4 and add JAR files to the library. The JARs we need are js, jsapi,sphinx4 and tags. Then we can import the Hello World demo jar provided by Sphinx. All this importing works is simple. One thing should be pointed out is that the default heap memory provided by JVM(Java Virtual Machine) is not big enough for Sphinx programs. So we must ask for more heap memory by typing the below command in Cmd command window:

After this, we can run the Hello World demo. Now let's have a look at the source of the Hello World demo. To run a program using Sphinx4, three files are required:

* The Java file to compile and run.
* The grammar file.
* The configuration file

The tasks of Sphinx java files are the same as common java files, show the logic of the programs and tell the JVM what to do. The grammar files restrict the interaction grammar that one can use in the application. For example, in this Hello World demo, the grammar is:

public <greet> = (Good morning | Hello) ( Bhiksha | Evandro | Paul | Philip | Rita | Will );

That means a sentence that can be recognized is consist of one of the two phrases within the first parenthesis and one of the six words within the second parenthesis. The configuration file is used in the Java file at the beginning to tell the Java file about the configuration information, including path to acoustic model, path to dictionary, and path to language model.

This Hello World demo has already set a good framework for us. Out Voice-Controlled calculator is based on this framework. The rest works we need to do are:

* Write the Java file of the Voice-Controlled calculator
* Write a new grammar which will be used in our program
* Make a new dictionary(The old acoustic model and language model are appropriate because the language used is English)
* Change the configuration file

## Java File Implementation

(1) Project Design

Basically, our program has two main parts: one is in charge of interacting with the user and presents the result graphically; The other one handles all works behind GUI. For instance, set the configuration file, do the computing, and receive and execute the input voice command. For our program voice is use for command and control, so it is a good chose to use a finite state grammar. All the design should take user experience into consideration.

(2)Logic of the program

Here is the process when people using a normal calculator. First, they type the number buttons from 0 to 9 to form the first executed number . Then they chose the kind of operation they want, usually addition, subtraction, multiplication and division. After that, typing the second operation number . Finally, press "=" button to get the result.

As for a Voice-Controlled calculator, we can just simulate the process of using a normal calculator. For example, when you say "One", the calculator acts as button 1 has been pressed. Then you say "Two", button 2 has been pressed and you get the number 12. You can keep going until you get the number you want. After you are done with input of the first number, you say "plus"(or whatever legitimate commands). Then you give the second number, same process as the first number. Finally, when you say "equal to", you get the result.

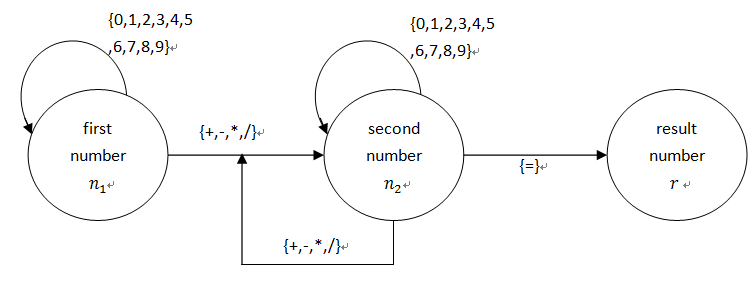


Fig.

We can see a strange arrow from Fig.1. There is an arrow represent operation signs {+,-,\*,/} goes out from the second number state. Can we still change the operation when we finish inputting the two numbers? For a normal calculator, we cannot. If you find out that you have inputted the two operating numbers right but inputted a wrong operation sign by mistake, unfortunately, you have to clear all the inputs and do it again. For the Voice-Controlled calculator, however, you can change the kind of operations you want at any time before the command "equal to(=)" without any effect on the operating numbers. That 's one of the advantages of a Voice-Controlled calculator. It's easy to understand. The inputs of a normal calculator is regarded as a whole part, while the input of the Voice-Controlled calculator is several separated parts.

An alternative option is to make the Voice-Controlled calculator more intelligent. For example, one say "One hundred and twenty four plus sixty one time five" and the calculator will return "124+61\*5=429". It seems a better choice because the calculator is smarter. However, the drawbacks are also significant. First of all, for example, you want the number "1234", it's quicker to say "One two three four" than "One thousand two hundred thirty four". Second, the recognition is not always accurate. Mistake happens especially when there is noise in the environment. And it is easy to modify single input number than changing the whole number.

Flow process diagram is shown as (Last page)

(3)GUI

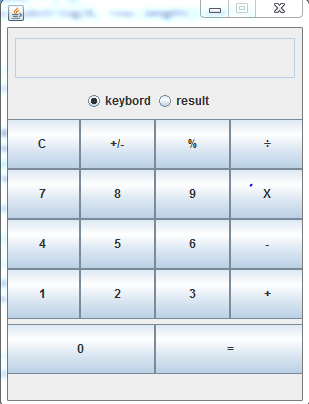


Fig.

The main frame of the calculator is shown as Fig.2. From this figure we can see the calculator has two frames, *keyboard* and *result*. We can chose which panel we want by voice command. Since it is a Voice-Controlled calculator, we even don't need a keyboard. However, this graphic stuff can give users better experience. When you say the buttons of group {0,1,2...9} or say the buttons of group {+,-,\*,/}, the corresponding button will change the color to red(like you keep pressing the button) and also appears on the display panel. This helps you to notice whether the recognition is wrong in a visual and direct way. Animations are different for button {C} and {=}. If you say button {C} or {=}, the corresponding button change its color to red for a second and then change its color back to normal(like you pressed the button once). The reason of the design of {C} and {=} is that these kind of commands will give the user a result immediately.

Of course, this keyboard is not necessary, so we have another frame *result.* The result panel is consist of a display panel which shows the current calculated number and a result panel which shows all the historical calculated results. In this frame, once you finished doing a calculation, the result will be stored and shown in the result panel.

## Grammar

The grammar format we used here is called Java Speech Grammar Format(JSGF). This website is a great tutorial of JSGF:

https://puneetk.com/basics-of-java-speech-grammar-format-jsgf

Here we can use two grammars. One is just list all the single command we need. It looks like:

public <calculator> = (one | two | three | ... | plus | minus | divided by | times | equals);

To say rule <calculator>, user must say "one" or "two" or... Only one of these words. This grammar is very easy and it is a simulation of the normal calculator. Each one word is just like pressing each one button.

We can also express it as a whole part. The grammar is:

public <calculator> = (one | two | three | ... | nine) + (plus | minus | divided by | times |)(one | two | three | ... | nine) +(euqals);

Any group or expansion followed by "+" indicates that it may be spoken one or more times. To say rule <calculator>, user must choose at least one word from the first number group. For example " One" or "One two". Then user chooses one of the four operators. After that, again, user say the second number group. Finally, the word "equals" is mandatory.

The advantage of using the second grammar is that it react very fast, because you save a great time on inputting. In this way you don't say one word, wait until it is recognized and then say another word. However, there are also two disadvantages for the second grammar. One is that if some part of the whole input is recognized wrong, you have to redo the whole input. The other drawback is that you have to give the input all in one time. You cannot stop and think during input.

## Dictionary

We will create the dictionary using *Sphinx Knowledge Base Tool*. This tool is used for building a set of lexical and language modeling files for Sphinx decoders.

To generate the dictionary, first we need to create a sentence corpus file, consisting of all sentences we would like the decoder to recognize. There should be just one sentence in a line. Then we upload the sentence corpus file. The website will compile the corpus file and generate the dictionary for us.

Here is the web site:

http://www.speech.cs.cmu.edu/tools/lmtool-new.html

# RESULT AND ANAYSIS

This Voice-Controlled calculator works very well in quiet place. However, when there is noise in the environment, such as people's talking or sound of vehicle engine, the recognition accuracy is unacceptable. In further improvement, I can work on how to filter the noise. And I'm not satisfied with the reaction speed of the calculator. That's also one thing need to be improved.

1. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
2. M. Young, The Technical Writer’s Handbook. Mill Valley, CA:

**Program Process**

yes

no

no

number

{0,1,...9}

operation sign

{+,-,\*,/}

{=}

recognize the voice and find it in the dictionary dictionary

match?

set up config file

set up microphone

get the input voice

return: wrong command

{Clear}

clear all

legitimate expression?

return result

error

first num1

second num2