Circuits and Systems, 2016, 7, 69-82
Published Online March 2016 in SciRes. http://www.scirp.org/journal/cs
http://dx.doi.org/10.4236/cs.2016.73008

Scientific Research Publishing

Real Time Speech Based Integrated Development Environment for C Program

Bharathi Bhagavathsingh, Kavitha Srinivasan, Mariappan Natrajan

Department of CSE, SSN College of Engineering, Kalavakkam, India
Email: bharathib@ssn.edu.in, kavithas@ssn.edu.in, marriappan.n@gmail.com

Received 14 February 2016; accepted 17 March 2016; published 23 March 2016

Abstract

This Automatic Speech Recognition (ASR) is the process which converts an acoustic signal captured by the microphone to written text. The motivation of the paper is to create a speech based Integrated Development Environment (IDE) for C program. This paper proposes a technique to facilitate the visually impaired people or the person with arm injuries with excellent programming skills that can code the C program through voice input. The proposed system accepts the C program as voice input and produces compiled C program as output. The user should utter each line of the C program through voice input. First the voice input is recognized as text. The recognized text will be converted into C program by using syntactic constructs of the C language. After conversion, C program will be fetched as input to the IDE. Furthermore, the IDE commands like open, save, close, compile, run are also given through voice input only. If any error occurs during the compilation process, the error is corrected through voice input only. The errors can be corrected by specifying the line number through voice input. Performance of the speech recognition system is analyzed by varying the vocabulary size as well as number of mixture components in HMM.

Keywords

Automatic Speech Recognition, Integrated Development Environment, Hidden Markov Model, Mel Frequency Cepstral Coefficients

1. Introduction

Speech is one form of communication used by the humans for exchanging the information. Each word that is spoken by the humans is created using the phonetic combination of vowel and consonant speech sound units. Speech processing is the study of speech signals and processing methods of these signals. The speech signals are usually processed in a digital representation. Speech recognition is the process of converting the speech signal

into human readable text. Nowadays speech recognition is used in variety of applications. People with disabilities can benefit from speech recognition programs. For individuals that are deaf or hard of hearing, speech recognition software is used to automatically generate a closed captioning of conversations such as discussions in conference rooms, classroom lectures. Speech recognition is also very useful for people who have difficulty using their hands, ranging from mild repetitive stress injuries to involved disabilities that preclude using conventional computer input devices. Our proposed system is developed to facilitate the visually impaired people or the person with arm injuries with excellent programming skills that can code the C program through voice input. The paper is organized into existing systems, proposed system, implementation and its performance. Literature related with the proposed systems is discussed in Chapter 2. Chapter 3 deals with the proposed frame work followed by the implementation in Chapter 4. The performance analysis is detailed in Chapter 5. Chapter 6 concludes with a few points as to the scope for future enhancement.

2. Literature Survey

Speech recognition [1] is used to convert the audio signals into human readable text format. Speech recognition is classified as two types according to number of users using it. Speaker dependent system—the system [2] recognizes the words only from the trained speaker. Accuracy of these systems is usually high. Speaker independent system—these systems [2] are able to be used by different individuals without training to recognize each person's speech characteristics. Speech recognition system is classified into two types based on input to the system. Isolated speech recognition-it operates on a single word at a time [3], requiring a pause between saying each word. Continuous speech recognition [4] [5]—It operates on speech in which, words are connected together. Speech recognition is classified as three types according to sub word unit [3] used for recognition. The types are: syllable-based Recognition, phoneme-based recognition and word-based recognition. There are two phases in developing speech recognition system. They are training and testing phase. In training phase, the speech samples are collected. The features are extracted from the collected speech samples and the acoustic model is built. During testing phase, using acoustic model speech utterance is recognized.

Different types of spectral features that [6] can be extracted during training phase are Linear predictive analysis (LPC), Linear predictive cepstral coefficients (LPCC), perceptual linear predictive coefficients (PLP), Mel-frequency cepstral coefficients (MFCC) etc. In our proposed work, MFCC features are extracted from the speech samples.

A few of the speech based applications developed are mentioned below.

In [7], a web based application to find out the user's mood from the emotions present in the speech signal is presented. It is used to overcome the difficulties in the present web education system's feedback. However this application will collect the feedback about the web page as voice samples. From the collected voice samples, it will identify the user moods. A client-server based speech recognition system is described in [8]. In this work, recognition is done at the server side. The client will transmit the speech signals to the server. Speech can be transmitted to the server through internet. The disadvantage of this approach is that the user cannot access these applications through low bandwidth connections. A speech browser is developed in [9]. This system is used to browse the worldwide web via speech input. Speech based e-learning is described in [10]. In [10], the speech recognition system uses client/server architecture. The client uses a Java applet, which is integrated in an HTML page. It takes the user's input and activates the corresponding service at the speech server.

3. Proposed Framework

Two major modules of the proposed framework are shown in **Figure 1**.

Module 1: Speech recognition.

Module 2: Building IDE for C program.

3.1. Speech Recognition

In Speech recognition training phase, feature vectors are extracted from the given speech signal. The extracted feature is used to build the acoustic model. In testing phase, from the test speech signal, the features are extracted. The extracted feature is compared with the acoustic model to produce the recognized text. Speech Recognition system is implemented using sphinx [11]-[13] tool kit.

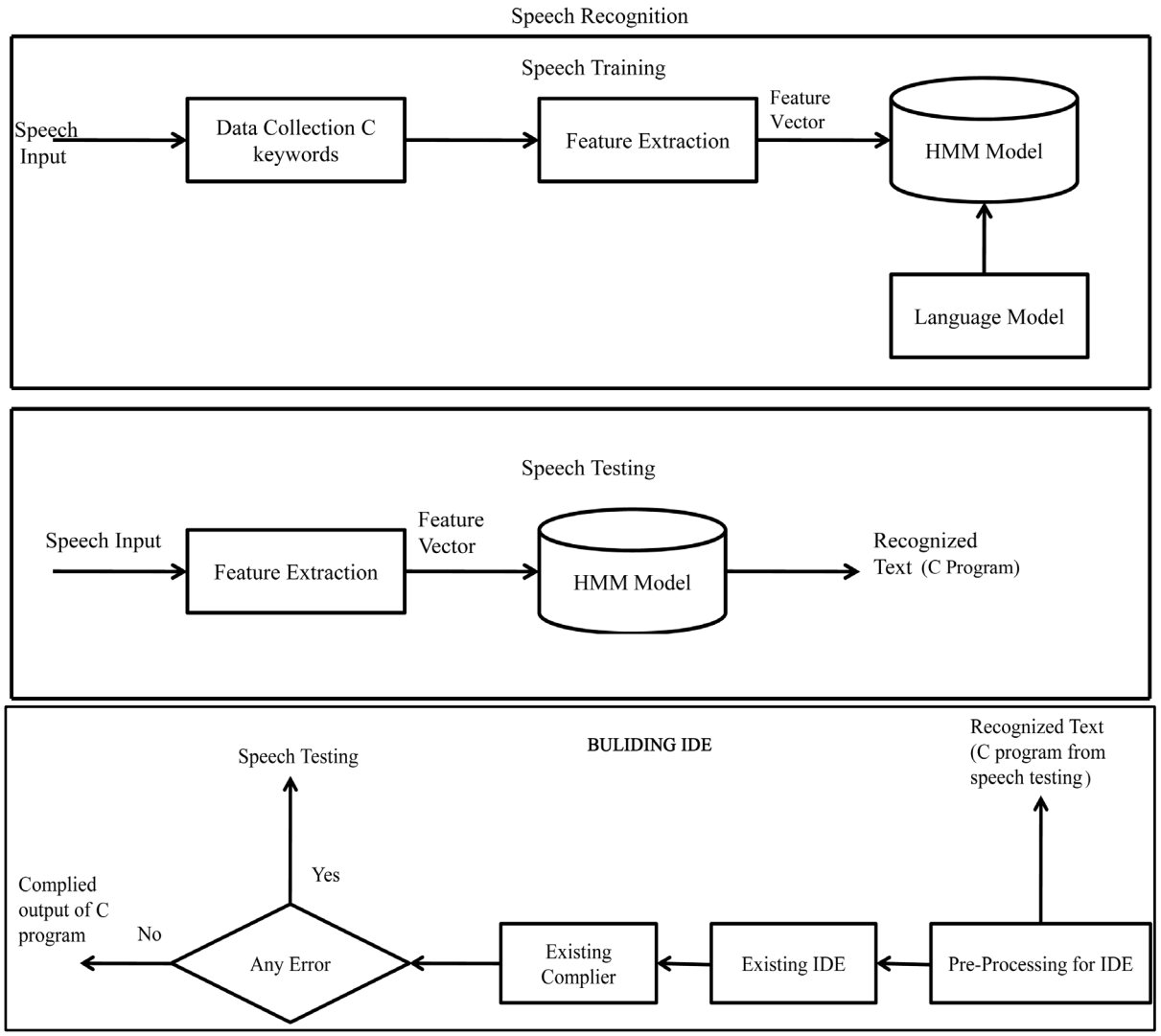

Figure 1. Proposed framework.

3.2. Building IDE for C Program

Recognized text from module 1 is pre processed to convert the text into proper C program using syntactic construct of the C language. This C program will be fetched as input to IDE. This IDE will produce the compiled output of the recognized C program.

3.3. Proposed Framework Description

3.3.1 Training Phase

The training phase consists of the following modules.

1) Data collection

Speech utterance correspond to C keywords are collected from different speakers. For each keyword, sixty speech utterances are collected. All speech samples are recorded in wav format. After collecting the voice samples, dictionary file, fileids, transcription files, language model files are generated. The fileids contains location of the wav file, transcription file contains the text corresponding to the wav file, dictionary file lists all the words and its phoneme sequence, language model file contains the probability of occurrence of each word in the speech corpus.

Example of dictionary file:

A AH
ADD AE D
AMBERSAND AE M B ER S AE N D

2) Feature Extraction

Features are extracted from the voice samples. MFCC features [14] [15] are extracted using the following steps shown in **Figure 2**.

Pre emphasis—Divide the signal into 20 - 40 ms frames. In this paper the frame size is assumed as 25 ms. This means the frame length for a 16 kHz signal is 0.025 × 16,000 = 400 samples. Frame step is usually 10 ms to 15 ms, which allows overlap between the frames.

Hamming windowing—Windowing is applied to minimize the disruptions at the start and at the end of the frame.

Fast fourier transform—The conversion from time domain to frequency domain is carried out by fourier transform method.

$$S_i(k) = \sum_{n=1}^{N} s_i(n) h(n) \mathrm{e}^{-j2\pi kn/N} \quad 1 \le k \le K \tag{1}$$

where $h(n)$: N sample long analysis window,

K: the length of the DFT.

The periodogram—based power spectral estimate for the speech frame $s_i(n)$ is given by

$$P_i(k) = \frac{1}{N} \left| S_i(k) \right|^2 \tag{2}$$

Mel Filter Bank Processing—The filters are used to compute a weighted sum of spectral components to filter the output.

Mel Scale—The Mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Humans are superior at discerning minute alterations in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features matches closely with humans' perception.

The formula for converting frequencies into Mel scale is:

$$M(f) = 1125 \ln\left(1 + \frac{f}{700}\right) \tag{3}$$

Discrete Cosine Transform—It is used to convert the Mel spectrum to the domain of time.

Delta Energy and Delta Spectrum—It is necessary to add features related to the change in the characteristics of cepstral over the time. Delta energy and delta spectrum are also known as differential and acceleration coefficients. The MFCC feature vector describes only the power spectral envelope of a single frame, however speech would also have information in the dynamics *i.e.* what are the trajectories of the MFCC coefficients over time. It turns out that calculating the MFCC trajectories and appending them to the original feature vector increases ASR performance by quite a bit.

Delta coefficients are computed as follows:

$$d_t = \frac{\sum_{n=1}^{N} n\left(c_{t+n} - c_{t-n}\right)}{2\sum_{n=1}^{N} n^2} \tag{4}$$

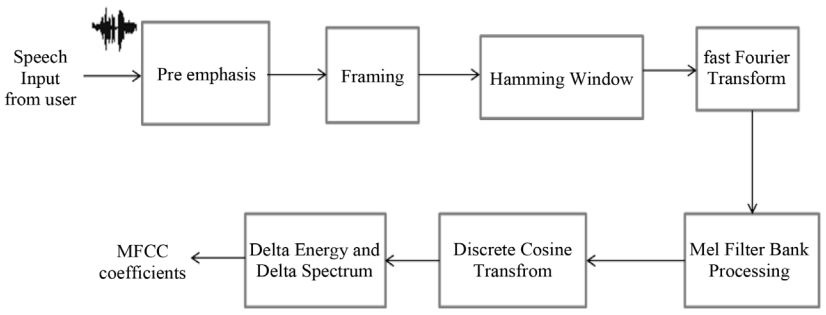

Figure 2. MFCC feature extraction steps.

where d_t is a delta coefficient, from frame computed in terms of the static coefficients c_{t+N} to c_{t-N}.

3) Building HMM model

A hidden Markov model (HMM) [16] is a statistical Markov model which has the unobserved states. The states in the Hidden Markov Model are not directly visible, *i.e.* it has hidden states. Each state has a probability distribution over the other states. Using HMM, the acoustic model is built from the extracted MFCC features. It is a word based model. Here each phone is represented as state. First state and last state are non-emitting state. The state transition between one phoneme to another phoneme. The state transition will lead to find out the vocabulary during the recognition process.

For example: INCLUDE IH N K L UW D

The word INCLUDE has 6 states. State transition from IH to D will lead to the word INCLUDE

4) Language Model

The language model is used to assign probability to each word according to their frequencies. This language model will facilitate to predict the subsequent word during the testing phase of the speech recognition. In speech recognition if the HMM model didn't predict the word correctly, then language model will find out the subsequent sequence word using these calculated probability values. Different types of language models can be built, e.g. unigram, bigram, trigram model, etc. Unigram model is used to find out one single word, whereas bigram model is used to predict the predecessor or successor word. Trigram model is used to predict the predecessor and the successor of the given word.

The unigram model can be calculated by:

$$P\left(w_1 w_2 \cdots w_n\right) = \prod_i P\left(w_i\right) \tag{5}$$

$$P\left(w_i\right) = \frac{C\left(w_i\right)}{N} \tag{6}$$

where $w_1 w_2 \cdots w_n$ represents the words in the corpus and $P(w_i)$ represents the probability of each word occurring in that corpus. i represents the i^{th} word in the corpus. $C(w_i)$ represents the count of the i^{th} word.

N represents the total number of words in the corpus.

The bigram model can be calculated by:

$$P\left(\frac{w_i}{w_1 w_2 \cdots w_{i-1}}\right) = \prod_i P\left(\frac{w_i}{w_{i-1}}\right) \tag{7}$$

$$P\left(\frac{w_i}{w_{i-1}}\right) = \frac{count\left(w_{i-1}, w_i\right)}{count\left(w_{i-1}\right)} \tag{8}$$

where $w_1 w_2 \cdots w_{i-1}$ represents the previous words in the corpus and $P\left(w_i / w_{i-1}\right)$ represents the probability of each word with the other words occurring in the corpus. i represents the i^{th} word in the corpus.

The trigram model can be calculated by:

$$P\left(\frac{w_i}{w_1 w_2 \cdots w_{i-1}}\right) = \prod_i P\left(\frac{w_i}{w_{i-2}, w_{i-1}}\right) \tag{9}$$

Example of language model:

(unigram)-3.8770 IF

(bigram)-3.3998 <s>OPENBRACKET

(trigram)-0.3010 <s>OPENBRACKET </s>

<s>: represents a word in the word corpus that occurs predecessor of the current word. </s>: represents a word in the word corpus that occurs a successor of the current word. Here probability values are represented in logarithm.

3.3.2. Speech Testing

1) Feature Extraction

The MFCC features are extracted from the test speech utterance. The procedure for extracting MFCC feature is explained in Section 3.3.1.

2) Acoustic Model

After extracting the MFCC features from test speech signal, using the acoustic model the text is recognized. In order to incorporate the syntax of the C program, the recognized text is given to the IDE Preprocessing module. Speech testing is implemented in two ways.

Real time speech testing-Voice input is given instantly to recognize the text.

Recorded wav file speech testing-The recorded wav files are given as input to this type of testing.

The algorithm for Real time speech testing is shown in algorithm 1. In algorithm 1, IDE commands represent open, save, new, compile and run. The algorithm for doIDEcommands function and doIDEpreprocessing function are explained in algorithm 3 and 6. The algorithm for recorded wav file speech testing is given in algorithm 2. The algorithms are explained in **Appendix**.

3.4. IDE Preprocessing

In IDE pre processing module, the recognized text from speech testing will be converted into C program using the syntactic construct of C language. In the first step, the recognized text is divided into tokens. If token is recognized as symbol then replace the token with its corresponding symbol. If token is a number then convert the token into its equivalent number. If the token is not a number or a symbol then leave the text as it is. After the recognized text is pre processed, it will be fetched as input to the IDE module.

Text to Symbol and Number Conversion

Create two look up tables for storing the symbols (operators in C language) and numbers. Compare the token with the symbols present in the look up table for symbols. If one of the symbols matches with the token then replace the token with its corresponding symbol. Otherwise, compare the token with the numbers present in the look up table for numbers. If one of the numbers matches with the token then replace the token with its corresponding value. If the token does not match with all of the symbols or numbers then leave the token as such. This process will be repeated for all the tokens. Few symbols and all numbers are listed in look up **Table 1** & **Table 2** respectively.

The algorithms for doing IDEpreprocessing are shown in algorithm 3, 4, 5.

3.5. Existing IDE

In this module, the pre processed text will be fetched as input. IDE commands are also given through voice input only. The IDE commands used in our proposed work are open file, save file, new file, compile file and run file.

3.5.1. New File

This command will open a new file in the IDE. The voice command will create a new file in the IDE.

Table 1. Look up table for symbols.

Key	Value
Less than	<
Equal to	=
Greater than	>
Open curly braces	{
Close curly braces	}
Dot	.
Open bracket	(
Close bracket	)
Hash	#
Backslash	\
Plus	+

Table 2. Look up table for numbers.

Key	Value
zero	0
One	1
Two	2
Three	3
Four	4
Five	5
Six	6
Seven	7
Eight	8
Nine	9

3.5.2. Open File

The open file command is used to open the existing file. For opening a file the user has to provide the command "open file <filename> dot c 'or' open file location <location of the file>" through voice input. If the file exists subsequently the IDE will open the file. If it is not exists in that case it will show an error message to the user.

If we utter the following lines:

Open file example dot c (or) open file location D colon backslash example backslash example dot c.

This will be converted into the text as follows:

Open file example.c

Open file location D:\example\example.c

The voice command for open file is converted into text. From the text, file name will be extracted. If the command is open file without the keyword "location" in that case it will look for the file in the current directory. If the command is with the keyword "location" then it will look for the file at the specific location.

3.5.3. Save File

The save command is used to save a file. For saving a file the user has to provide the command "save file <file-name> dot c 'or' save file location <location of the file>" through voice input. It will save the contents of the IDE into the file specified by the user.

Example: save file example dot c (or) save file location D colon backslash example backslash example dot c.

These will be converted into the text.

Save file example.c

Save file location D:\example\example.c

The voice command will be converted as text. From the text the

File name will be extracted. If the command is save file without keyword "location" then it will save the

File in the current directory. If the command is with the keyword "location" then it will save the file at the specific location.

3.5.4. Compile File

The compile command is used to compile the C file. For compiling a file the user has to provide the command "compile file" through voice input. The file is compiled using the gcc compiler. After the compilation process the gcc compiler will produce the .exe file or .o file based on the Operating System. It will produce .exe files for windows and .o file for unix based Operating System.

3.5.5. Run File

The run command is used to run the C file. For running a C file the user has to provide the command "run file" through voice input. If the .exe file or .o file is found then it will run the .exe file or .o file in the command prompt or terminal according the Operating System. If the .exe file or .o file is not found then it will display the error message. After running the C program the .exe or .o file is deleted.

3.5.6. Goto Line Number

The goto line number command is used to correct the errors occurred during the compilation of the C program. For error correction in a C file, the user has to provide the command "goto line number <lineno>" through voice input. Extract the line number from the user voice command. Empty the text in that line. After clearing the text in the specific line given by the user, place the latest recognized text. The new text is recognized from the user voice.

Example: goto line number six six.

From this six six should be extracted and converted to numbers as 66. This text to number conversion will be done by the IDEP reprocessing module. The algorithm for doing IDECommands is shown in algorithms 6, 7, 8, 9, 10.

4. Experimental Setup

We have collected 217 C programming language keywords. Speech utterances corresponding to these keywords are collected from 25 speakers. Each keyword is uttered 20 times. The speech samples are collected using microphone. Speech data is decoded with sampling rate of 16 KhZ with single bit mono channel stored in WAV format. After the data collection, transcription file, dictionary file and language model files are generated. From the collected speech samples, the MFCC features are extracted. Using the extracted features, HMM model is built. During testing, using the HMM model, the test utterance is recognized. In IDEpre processing, recognized text is converted into C program using syntactic construct of the C language. The IDE commands open file, save file, compile file, run file and goto line number also implemented using the voice input.

5. Performance Analysis

The performance measure used to evaluate the proposed system is discussed below.

Performance Measures

The Word Error Rate (WER) is a metric which is used to measure the performance of an ASR. It compares the given word to a recognized word and is defined as follows:

$$WER = \frac{S + D + I}{N} \tag{10}$$

where:

S is the number of substitutions,

D is the number of deletions,

I is the number of insertions and,

N is the number of words in the actual word.

Word Error Rate calculation for the entire system is as follows:

$$WER_{\text{system}} = \frac{WER_{\text{cprogram}} + WER_{\text{idecommands}}}{2} \tag{11}$$

Word Error Rate calculation for a C program is

$$WER_{\text{cprogram}} = \frac{\sum_{i=0}^{n} WER_i}{n} \tag{12}$$

where:

i is the line number of the program,

$WERi$ is the Word Error Rate for the i^{th} line of the C program,

n is the total line numbers in a C program.

Word Error Rate calculation for IDE commands is

$$WER_{\text{idecommands}} = \frac{\sum_{c=0}^{m} WER_c}{m} \tag{13}$$

where:

c is the voice input for IDE command,

WERc is the Word Error Rate for the IDE command *c*,

m is the total no. of IDE commands uttered by user.

The performance of speech recognition is analyzed by varying the number of mixture components using 150 words are tabulated in **Table 3**. From **Table 3**, it has been noted that word error rate is decreased when number of mixture component is increased. There is an increase in word error rate for mixture component 128. The reason for increase in word error rate is that, the amount of training data is not sufficient to train the model for 128 mixture component. The number of times the word is uttered is increased to 40 to improve the performance for 128 mixture component. The number of mixture component in HMM model is decided based on the number of phonemes available in the training utterances. From the above experiment, it has been concluded that the optimum number of mixture component for the current study is fixed as 64 components.

The performance of speech recognition is analyzed by varying number of words are represented in **Figure 3**. From **Figure 3**, it has been noted that word error rate is increased when the vocabulary size is increased. For large vocabulary speech recognition system, the suitable sub word unit is phoneme or syllable.

6. Conclusion

Our proposed system is used to capture the C program through voice input and produces the compiled C program as output. During training phase, speech utterances corresponding to C key word are collected. MFCC features are extracted from the speech samples. HMM model is built using extracted features. During testing, from the test utterance, the MFCC features are extracted. Using the HMM model, the text is recognized. The recognized text is converted into the C program by using syntactic constructs of the C language. The IDE commands for saving, opening, compiling and running the file are also given through voice input. The proposed speech based IDE is implemented for C program only, it can be extended to other programming languages. In our proposed work, word based speech recognition is implemented. While extending the research work to other programming languages, phoneme based speech recognition can be applied. Phoneme based speech recognition supports the large vocabulary data set.

Table 3. Performance of speech recognition system using different mixture components.

No. of mixtures	Word error rate (in %)
2	58.8
4	42.0
8	27.5
16	13.0
32	4.6
64	3.8
128	9.2

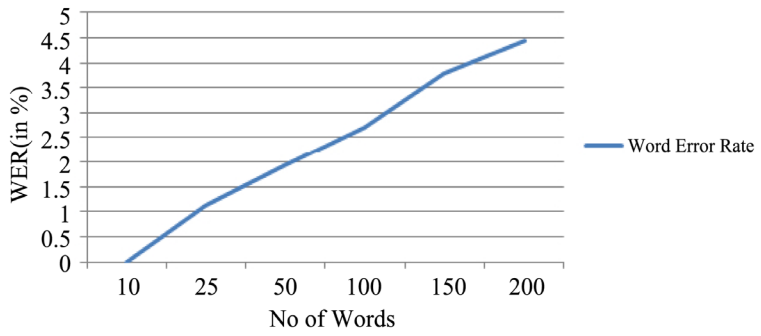

Figure 3. WER vs. number of words.

References

[1] Ben Mosbah, B. (2006) Speech Recognition for Disabilities People. *Information and Communication Technologies*, 2006, Vol. 1, 864-869. http://dx.doi.org/10.1109/ictta.2006.1684487

[2] Huang, X.D. and Lee, K.F. (1993) On Speaker-Independent, Speaker-Dependent, and Speaker-Adaptive Speech Recognition. *IEEE Transactions on Speech and Audio Processing*, **1**, 150-157. http://dx.doi.org/10.1109/89.222875

[3] Yusnita, M.A., Paulraj, M.P., Yaacob, S., Abu Bakar, S., Saidatul, A., Abdullah, A.N. (2011) Phoneme-Based or Isolated-Word Modeling Speech Recognition System? An Overview. 2011 *IEEE* 7*th International Colloquium on Signal Processing and Its Applications* (*CSPA*), Penang, 4-6 March 2011, 304-309. http://dx.doi.org/10.1109/CSPA.2011.5759892

[4] van der Walt, C. and Mortimer, B. (1993) The Practical Application of a Continuous Speech Recognition System. In *Proceedings of the* 1993 *IEEE South African Symposium on Communications and Signal Processing*, Jan Smuts Airport, 6 August 1993, 145-149. http://dx.doi.org/10.1109/comsig.1993.365855

[5] Ganapathiraju, A., Hamaker, J., Picone, J., Ordowski, M. and Doddington, G.R. (2001) Syllable-Based Large Vocabulary Continuous Speech Recognition. *IEEE Transactions on Speech and Audio Processing*, **9**, 358-366. http://dx.doi.org/10.1109/89.917681

[6] Shrawankar, U. and Thakare, V.M. (2013) Techniques for Feature Extraction in Speech Recognition System: A Comparative Study. *International Journal of Computer Applications in Engineering, Technology and Sciences*, **2**, 412-418.

[7] Gong, M.M. and Luo, Q. (2007) Speech Emotion Recognition in Web Based Education. In *IEEE International Conference on Grey Systems and Intelligent Services*, Nanjing, 18-20 November 2007, 1082-1086.

[8] Digalakis, V.V., Neumeyer, L.G. and Perakakis, M. (1999) Quantization of Cepstral Parameters for Speech Recognition over the World Wide Web. *IEEE Journal on Selected Areas in Communications*, **17**, 82-90. http://dx.doi.org/10.1109/49.743698

[9] Borges, J.A., Jimenez, J. and Rodriquez, N.J. (1999) Speech Browsing the World Wide Web. In 1999 *IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 4, 80-86. http://dx.doi.org/10.1109/icsmc.1999.812380

[10] Werner, S., Wolfi, M., Eichner, M. and Hofimann, R. (2004) Integrating Speech Enabled Services in a Web-Based E-Learning Environment. *International Conference on Information Technology*: *Coding and Computing*, Vol. 2, 303-307. http://dx.doi.org/10.1109/itcc.2004.1286651

[11] Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P. and Woelfel, J. (2004) Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Technical Report.

[12] Lee, K.-F., Hon, H.W. and Reddy, R. (1990) An Overview of the SPHINX Speech Recognition System. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **38**, 35-45. http://dx.doi.org/10.1109/29.45616

[13] Huang, X.D., Alleval, F., Hwang, M.-Y. and Rosenfeld, R. (1993) An Overview of the SPHINX-II Speech Recognition System. *Proceedings of the Workshop on Human Language Technology*, Vol. 38, 81-86. http://dx.doi.org/10.3115/1075671.1075690

[14] Putra, B. (2011) Implementation of Secure Speaker Verification at Web Login Page Using Mel Frequency Cepstral Coefficient-Gaussian Mixture Model (MFCC-GMM). 2011 2*nd International Conference on Instrumentation Control and Automation* (*ICA*), Bandung, 15-17 November 2011, 358-363.

[15] Martine, J., Perez, H., Escamillal, E. and Suzuki, M.M. (2012) Speaker Recognition Using Mel Frequency Cepstral Coeficients (MFCC) and Vector Quantization (VQ) Techniques. In 2012 22*nd International Conference on Electrical Communications and Computers* (*CONIELECOMP*), Cholula, 27-29 February 2012, 248-251.

[16] Rabiner, L. (1989) A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, **77**, 257-286. http://dx.doi.org/10.1109/5.18626

Appendix

Input: Feature file corresponding to voice input, Grammar file, HMM model, dictionary file
Output: Recognized text - C program or IDEcommands
Variable: *text, recognizedText;*
while *true* do
recognizedText <- Recognize the text from voice input;
if *recognizedText* == "Close IDE" **then**
exit;
end
else if *recognizedText* == IDE*commands* **then**
doIDEcommands*(recognizedText);*
end
else
text ← do IDEPreprocessing(*recognizedText) ;*
place the *text* in IDE;
end
end

Algorithm 1: Algorithm for Real time speech testing

Input: Featue file corresponding to each line of the C program, grammar file, HMM model, dictionary file
Output: Recognized Text
Variable: *text, recognizedText;*
while *read the wav file one by one until last file in the directory* **do**
recognizedText ← Recognize the text from wav file;
if *recognizedText == IDEcommands* **then**
doIDEcommands*(recognizedText);*
end
else
text ← do IDEPreprocessing*(recognizedText) ;*
place the *text* in IDE;
end
end

Algorithm 2: Algorithm for recorded wav file speech testing

Input: recognized text
Output: Preprocessed Text
Function doIDEPreprocessing()
Array A[ ]←split the recognized text into tokens;
let p ← List of tokens;
readfi token ← read first token;
while A*[read token]* <=|p| **do**
if *isSymbol(read token)* **then**
convert the token into symbols;
end
else if *isNumber(read token)* **then**
convert the token into numbers;
end
else
don't change token ;
end
read token ← read the next token;
end
return *[concatenate all the tokens to form the string];*

Algorithm 3: Algorithm for doIDEPreprocessing

```
Input: token
Output: return true if the token is symbol, else return false
Function isSymbol()
      if token is found in the symbol table then
          return true;
      end
      else
          return false ;
end
```

Algorithm 4: Algorithm for isSymbol

```
Input: token
Output: return true if the token is number, else return false
Function isSymbol()
     if token is found in the number table then
          return true;
    end
    else
          return false ;
  end
```

Algorithm 5: Algorithm for isNumber

```
Input: Recognized text-IDE commands
Output: Execute the IDE commands
Function doIDECommands()
     switch IDEcommand do
          case new file
               Empty the contents in the IDE ;
               Creates a new file ;
               break ;
          end
          case open file
               openFile(recognizedText) ;
               break ;
          end
          case save file
              saveFile(recognizedText) ;
              break ;
          end
          case compile file
               compileFile() ;
               break ;
          end
          case run file
              runFile() ;
              break ;
          end
          case goto line number
              gotoLine() ;
              break ;
          end
  end
```

Algorithm 6: Algorithm for IDECommands

```
Input: Recognized text
Output: save the file
Function saveFile()
    Variable: fileName;
     if command == "save file" then
        after the keyword "file" take the text as string;
        fileName ← doIDEPreprocess(string);
        save the file using bufiered writer;
     end
     else if command =="save file location" then
            after the keyword "location" take the text as string;
            fileName ← doIDEPreprocess(string);
            save the file using buffered writer;
       end
```

Algorithm 7: Algorithm for save file command

```
Input: Recognized text
Output: Open the file
Function openFile()
Variable:fileName;
if command == "open file" then
  after the keyword "file" take the text as string;
  fileName ← doIDEPreprocess(string);
   if fileExists(fileName) then
     open the file using bufiered reader;
  end
else
     display error message ;
end
end
else if command == "open file location" then
       after the keyword "location" take the text as string;
       fileName ← IDEpreprocess(string);
       if fileExists(fileName) then
         open the file using bufiered reader;
       end
else
         display error message ;
end
end
```

Algorithm 8: Algorithm for Open file command

```
Output: Compile the file
Function compileFile()
    copy the contents from the IDE and save the file as temp.c;
    compile the file using gcc compiler ;
if compilation is successfull then
    produce the temp.exe file ;
    notify the user with a message;
    delete the temp.c file ;
end
else
     display compilation error to the user ;
end
```

Algorithm 9: Algorithm for Compile file command

Output: Run the file
Function runFile()
if *fileExists(temp.exe)* **then**
    run the temp.exe file in the command prompt ;
    display the output in the IDE ;
    after run delete the temp.exe file ;
end
else
    display error message to the user ;
end

Algorithm 10: Algorithm for Run file command

Input: Recognized text
Output: goto the line no and place the new recognized text
Function gotoLine()
    Variable:lineNo;
    after the keyword "Line Number" take the text as string;
    lineNo <- IDEpreprocess(string);
    place the cursor in the lineNo in the IDE;
    remove the text in that specific lineNo in the IDE ;
    recognize the text from the gotoline.wav file ;
    place the recognized text on to the specific line of the IDE ;

Algorithm 11: Algorithm for Goto Line Number command