Voice Activation Software Documentation

Author: Yukti Suri

Date: 22/09/14

Introduction

All medical robotic systems as well as laparoscopic procedures make use of endoscopic systems which provide the internal high definition view to surgeons. As surgeons work with tools, the camera is held and maneuvered by an assistant either manually or using robotic manipulators. Having a voice controlled robotic camera holder gives the surgeon the complete and easy control of the entire procedure. With this objective in mind, AVRA has endeavored to produce a software which would allow for speech recognition and translating those voice commands into specific and safe camera holder robotic manipulator movements.

Platform and Software Dependencies

This software is developed using Qt 5.2.1 (C++) and can run on both Windows and Linux. All the required static libraries have been attached in the parent folder under '*third party*' subfolder.

The software basically makes use of Pocketsphinx library from CMU Sphinx Speech Recognition Project (http://cmusphinx.sourceforge.net/). This is an open source speech recognition toolkit which has been used extensively in various research as well as commercial projects.

Code Directory Structure

The code directory named as "*VoiceActivationVerN*" is the parent directory for the source code and various dependencies. The project requires pre-installation of CMU Sphinx libraries (sphinxbase, sphinxtrain and pocketsphinx) in a folder named "*CMUSphinx*" in the same folder as VoiceActivationVerN. The installation and build instructions for Sphinx tools can be found here. The folder structure should be as follows:

```
Parent_Folder (any name)
        |
        |_____--> CMUSphinx
        |                 |
        |                 |_____-->pocketsphinx
        |                 |_____-->sphinxbase
        |                 |_____-->sphinxtrain
        |
        |_____-->VoiceActivationVerN
                          |
                          |_____--> bin
                          |_____--> build
                          |_____--> include
                          |_____--> models
                          |_____--> src
                          |_____--> thirdparty
```

The subfolder *bin* is the target for the project executable. This is also the folder path from where the executable should be run from command line. It contains all the .dll files required to run the executable from command line (for windows).

The subfolder *build* contains all the config and make files. It is the folder which contains the project's QtCreator .pro file which should be opened to open the complete project in QtCreator.

The subfolder *include* contains the header files of the project.

The subfolder *models* contains all the adapted acoustic models in user defined separate folders. The *defaultModel* is the folder which is used as the basic structure for all the adapted models. The corpus.txt file contains the user defined speech commands in separate sentences. Sphinx lm tool (or language model tool) was used to generate model.lm, model.log_pronounce, model.sent and model.vocab. Along with these files the default acoustic model, namely "*hub4wsj_sc_8kadapt*" is part of the folder. For our application, as the command keyword set would remain unchanged, all these files and folders are copied into each new user defined model e.g. "testModel". These are used for model adaptation, which is also a feature of this software.

The subfolder *src* contains all the source files of the project.

The subfolder *thirdparty* contains various applications which are used in the source codes.

Voice Activation Project Modules

Following are the main software modules (classes) of the project:

VoiceActivationVerN
|
|_____-->SpeechTraining
|_____-->TcpClient
|_____-->VoiceBase

SpeechTraining Class

This class is the primary class which is used for the training for adaptation of acoustic models. This adaptation is required to enhance the accuracy of recognition, but it is not a full fledged training of a new acoustic model.

Some basic concepts required to understand the functionality of this class:

1) Corpus file: This is the name of the corpus file (default corpus.txt) which contains the keywords to be recognized in separate lines. For our application it is advisable to run the training module which would copy the default corpus file from *defaultModel* folder in *models* folder.

2) Base Audio File Name: According to the keywords in corpus.txt the user has to record audio files for training of module. For each line in corpus file a single audio file is generated. The default base name is used for all audio files unless specified otherwise.

Figure SpeechTraining Class Member Functions

The different member functions of this class have been described below:

1) void setPath(QString)
This function is used to set the working directory.

2) **QString getPath()**
This function returns the working directory path.

3) **void setBaseAudioFileName(QString)**
This function is used to set the base name of all audio files. This is optional. Default names are set as “file001.wav”, “file002.wav” and so on.

4) **QString getBaseAudioFileName()**
This function returns the base name of audio files. The default base name is “file”.

5) **int getNumberOfAudioFiles(QString)**
This function returns the total number of audio files in the folder. This should be equal to the number of commands or sentences in corpus file.

6) **void setCorpusFile(QString)**
This function is used to set the corpus file name. The default name (if this function is not called with another name) is “corpus.txt”.

7) **void startRecordandSave()**
This is the main function used to record the audio files as per the corpus.txt file. It also generates additional supporting files. The procedure is:

- Update path and model name.
- Read lines from *model.sent* file in the folder, one at a time

- Write the line in *model.transcription* followed by the auto incremented audio file name: file001, file002 etc
- Write the audio file name in *model.fileids* (one sentence is one filename): file001, file002 etc
- Display the read keyword command line for user to read out which is saved in file00n.wav file, where n is incremented in loop. **RECORD AT A SAMPLING RATE OF 16 KHZ IN MONO WITH SINGLE CHANNEL.**
- Repeat from second step till the end of *model.sent* file.

8) **void setModelName(QString)**

This function is used to set the model name. If this function is not called the default name is set to "model". All the supporting files would then be named as: *model.sent*, *model.fileids, model.transcription, model.lm, model.dic* etc.

9) **void updatePathModel()**

This function is used to update the model name using the path specifies. It uses the current value stored in private variable *path* and looks for the *.lm file in that location. The base name is automatically set as model name.

10) **void makeMFCFiles()**

As the name suggests this function is used to generate set of model feature files (*.mfc) from the WAV audio recordings. This is accomplished with sphinx_fe tool from SphinxBase. For the acoustic model used in this application the default parameters are already set and only a call to this function is required to generate all the mfc files.

11) **void collectStatistics()**

This function is required to accumulate observation counts from the adaptation data created. It creates gauden_counts, mixw_counts and tmat_counts files in the working directory. This function uses bw.exe tool from SphinxTrain to accomplish this. Again to simplify all the parameters for this particular application have already been set.

12) **void makeNewAdaptationModelUsingMap()**

First the default acoustic model is copied into the current model directory.
Using Maximum a Posteriori adaptation approach the parameters such as means, variances, mixture weights and transition matrices are updated in the acoustic model of the current model directory according to the collected statistics. For this map_adapt application is used from SphinxTrain library.

13) **void testNewModel (QString acousticModelFolder, QString testSetFolder)**

This function is used to calculate WER or Word Error Rate for the adapted model. This function takes in the adapted acoustic Model Folder Path as the first argument and test folder path (which contains the test wav files and transcription file) as the second argument. It runs the batch pocketsphinx recognition on the test wav files using the adapted acoustic model and generates a hypothesis file. A perl script is then run to compare the hypothesis file with the test transcription file and calculate the percentage accuracy.

14) **void runCmdProcess(QString cmd, QString currDir)**
This command is used to run the command line instructions as separate QProcess. The command is taken as a QString and second argument is used to set the working directory. First the path is changed and then the command is executed. Only minimal output is shown on the output console.

15) **void runCmdProcessWithOutput(QString cmd)**
This function executes the command passed as argument in the current working directory as set in the private variable *path* and shows the complete output on the output console.

Basic Adaptation Procedure Snippet

```
/**
 * ******************Adaptation******************/
/**
SpeechTraining set;
QString newModelName("YuktiModel");
set.setPath(commonModelPath + newModelName + "\\");

set.updatePathModel();

set.startRecordandSave();

set.makeMFCFiles();
set.collectStatistics();
set.makeNewAdaptationModelUsingMap();

/**
 * ******************Testing****************
  *Acoustic Model Folder = folder containing adapted model to be tested
  *Test Set Folder = folder containing test wav and analogous
  * transcription files to be used for testing
**/
QString acousticModelFolder = newModelName + "\\";
QString testSetFolder = "adaptedModel\\";
set.testNewModel(acousticModelFolder, testSetFolder);
```

TcpClient Class

This class provides the basic functionality for creating a client for TCP/IP communication. It creates a client object which connects to the server and sends the messages to it. In our application we use this class to send the recognized voice commands to the motion controller main server whenever a new hypothesis is generated. The motion controller would then use the motion planner to create robot movements based on the recognized commands.

Figure TcpClient Class Member Functions

The details of all the member functions of this basic utility class are given below:

1) **void start (QString address, quint16 port)**

This function takes the Host IP address and port as the two arguments and tries to connect with it if the server is running. It waits for 3000 ms to establish a connection. It sends an error message if the client is unable to connect to the server.

2) **void setMessage(QString)**
This function is used to set a *message* (a private data member) of the client object. The message is a QString. In our application these messages are the valid recognized voice keywords.

3) **QString getMessage()**
This function is used to retrieve the data saved in the private member *message* of the client object.

4) **void sendMessage()**
This function is used to send the value in *message* variable to the server. It waits till the message is sent and then flushes the client object.

5) **QAbstractSocket::SocketState getState()**
This returns the current state of the client object. The states are defined by the enum SocketState as: Unconnected State, Host Lookup State, Connecting State, Connected State, Bound State, Listening State and Closing State.

6) **void stop()**
This function when called closes the connection and destroys the client object.

VoiceBase Class

This class provides the necessary base for continuous speech recognition. This class is used to initialize basic parameters, set up the connection with the microphone and initiate listening mode for continuous speech recognition. A hypothesis is printed on the console output of the recognized keywords for each utterance (an utterance is defined as a speech snippet separated by relatively long pauses) and the validated hypothesis is sent over the TCP to the server.

Figure VoiceBase Class Member Functions

The details of all the member functions of this class are given below:

1) **void connectMicrophone ()**
As the name suggests before recognition the default microphone device is opened using this function.

2) **void initListening()**
This function initializes the continuous listening module and throws errors if either initialization of voice activity detection fails, or recording fails or calibration of voice activity detection fails at startup.

3) **void recognizeFromMic()**

This is the core function which runs a continuous loop to listen for any utterances. After each long pause a detected utterance is sent to the recognizer with the set parameters to decode and recognize the utterance. The found hypothesis is checked against a keyword command dictionary and sent over the TCP to the main server if found valid. If the speech command is recognized as "SLEEP" then the recognition is stopped otherwise the process of listening for next utterance resumes.

4) **int isHypValid()**

Checks the recognized keyword hypothesis against valid keyword dictionary of our application and returns true if a valid match is found. Only if the hypothesis is valid will the recognized keyword command is sent to the motion controller server.

5) **void sleepMs(int32 msec)**

This function is a basic timer utility to stall the compiler for specified number of milliseconds.

6) **int continuousVoiceRecognition(int argc, char* argv[])**

This is the main public function which is called along with user defined command line parameters. These arguments are parsed and accordingly the speech recognition parameters are set. If no error is found, then this function calls recognizeFromMic() function. This function returns a non-zero value in case an error is found.