Snackbot: The Process to Engage in Human-Robot Conversation

Dekita Moon[1], Paul Rybski[2], Cheryl Swanier[1], and Chutima Boonthum-Denecke[3]

[1]Fort Valley State University, Fort Valley, GA, 31030 USA

[2]Carnegie Mellon University, Pittsburgh, PA, 15213 USA

[3]Hampton University, Hampton, VA, 23668 USA

dekitamoon@yahoo.com, prybski@cs.cmu.edu, swanierc@fvsu.edu, chutima.boonthum@gmail.com

Abstract

While delivering snacks, Snackbot's need to actively engage in conversation with the customers and other individuals, provides an approach for verbal interaction. This paper addresses the verbal human robot interaction between humans and robots using a speech recognizer named Sphinx 4. Sphinx 4, written entirely in Java is capable of recognizing predetermined words and sentences. Thereby, allowing robots to actively engage in conversations using spoken language.

Introduction

In order to better serve customers during face-to-face encounters, verbal communication should be utilized during any transaction. Therefore, the need for Snackbot (see Figure 1, Lee et al. 2009) to communicate with humans requires implementing a way for a robot to recognize words using a speech recognizer. Speech recognition technologies allow computers equipped with microphones to interpret human speech, e.g. for transcription or as a control method. As of 2001, developers recognized the need to implement speech technology, capturing continuous speech with a large vocabulary at normal pace with an accuracy of 98% (Kumar, 2008). There is a huge commercial market waiting for this technology to mature, although there are few appliances using speech recognition as a method for communication and control. (Images SI Inc., n.d.).

Currently, Snackbot's speech is manipulated through the use of a dialog tree. This dialog tree can be used to easily create and manipulate nodes and transitions according to the previous interactions with customers. However, this dialog must be manipulated by a research assistant listening in on the conversation between Snackbot and its customer.

Figure 1. Snackbot (CMU, snackbot.org)

In this work, we aimed to develop a more autonomous way for Snackbot to acknowledge the responses from customers versus the research assistant taking on the task.

Methods and Materials

Before using a speech recognizer, you must be knowledgeable of what might be spoken between the customer and the robot. Speech recognizers, like those used on a daily basis, recognize speech that is predetermined by the developer such as automated phone systems for obtaining optimal performance. Likewise, being familiar with how the customer and Snackbot interact with each other allows the developer to train the application to look for certain inputs spoken. With

Snackbot initiating conversation with the customer, it is fairly easy to recommend what Snackbot will speak. Snackbot's dialog legend is a script in which it speaks to initiate conversation with its customers. The dialog legend encompasses many subjects that are categorized by the topic of conversation. The topics used by Snackbot include greeting, weather, gaining feedback from customers, and the offering of a snack.

However, Snackbot deals with many customers that are different. Therefore, customer's responses may vary significantly but are very dependent upon the predetermined dialog of Snackbot. For example, if Snackbot enters a room and asks for someone by name, there are several preconceived and data supported responses that might be made. Snackbot will recognize these responses and reply. To become familiarized with the responses that will be received, Snackbot's customers were recorded. These recordings were put in writing to be analyzed and categorized relevant to Snackbot's dialog legend. The categorization of the responses made by customers makes it easier to program Snackbot to look for certain phrases at specific times and respond accurately. Once the sentences used by customers were categorized, a training data set was made.

A training set is a text file listing all possible input spoken by customers as well as the possibilities that could have been said. For example, if a one customer states, "I want a cookie," it is important to list the potential alternatives to other snacks used in this sentence e.g. "I want a banana," "I want an orange," etc. Though tedious, forming the training set is one of the most important steps in using a speech recognition tool. Figure 2 (Li, 2005) shows a block diagram of two-stage processing in speech recognition: feature extraction and classifier (or speech recognizer). After the training set is made Sphinx-4 tools can be used to make a language model and plug in the appropriate data. With these tools, a training set can be used to create an N-gram. According to Jurafsky and Martin (2006), an N-gram is a probabilistic model, which predicts the next word from the previous N-1 words.

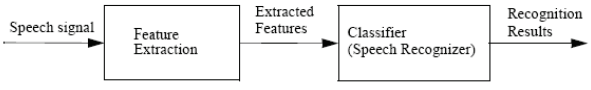

Figure 3. Two-stage processing in Speech Recognition

Sphinx-4 Tool

The Sphinx Knowledge Base Tool (CMU, n.d.; CMU, 2011) was used to convert the training set into a language model by simply choosing the training set text file and clicking compile knowledge base. Once compiled, the tool will create three documents including an N-gram language model, sentences, and dictionary to be used. In order to use these documents, Sphinx-4 should be downloaded from sourceforge.net. Sphinx-4 also requires JAVA SE 6 Development Kit (JDK), Ant 1.6.0, and Subversion (svn). Links to this necessary software are provided on the official website of Sphinx-4. The JDK tool makes your code able to compile. The Apache Ant is a command-line tool that is used to drive processes described in build files as targets and extension points dependent upon each other. The Subversion tool is recommended but is not required, allowing you to interact with the svn tree. An Integrated Drive Electronics (IDE) is also needed to build jar files used to compile and run Sphinx-4. There are several IDE tools that can be used but I choose to use Eclipse IDE for Java Developers.

Sphinx-4 lists several instructions on setting up your IDE, adding source folders, jar files, and build.xml ant files from Sphinx-4 to your project. It is important to note, Sphinx-4 also includes demos that includes its own language model or grammar, class, Manifest, and xml files. These files can be manipulated to cater to the needs of your project. The LiveConnectDemo in particular, provides an example of how to embed ECMAScript logic into JSGF grammars that will cause objects in your application to be activated when a JSGF RuleParse is processed by the edu.cmu.sphinx.tools.tags.ObjectTagsParser. If the demo is run and the speaker utters, "I want a pizza with mushroom and onions," Sphinx will recognize the order and perform the action tag designated with the utterance resulting in the textual feedback of, "pizza with mushroom and onions."

It is very important to make sure all of the words in the language model are included in the acoustic model. The acoustic model is a dictionary used to pronounce words within the language model using Alphabet symbols. Most projects will use the cmudict.0.6d file located in the WSJ 8gau 13dCep 16k 40mel 130Hz 6800Hz.jar.

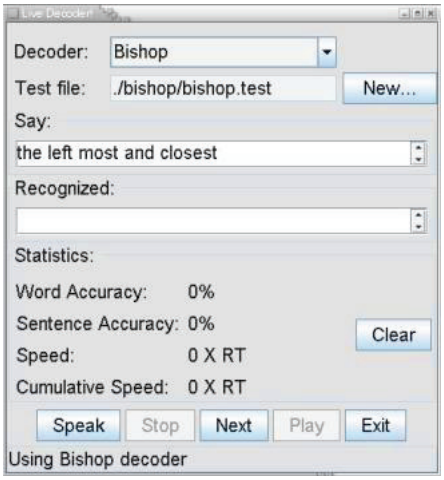

Figure 2. Sphinx 4 Live Demo Interface (CMU, 2011)

The acoustic model used within Sphinx 4 already provides a long list of word pronunciation or phonemes that may or may not include words needed for your individual purposes.

CMU (2011) provided a live test for Sphinx-4 (see Figure 3) using a sample test file, shown in Figure 4.

```
the left most and closest
the left most purple
the green one right in the middle
it's the uppermost of three in a row
this is a green one
this is the first behind
the one which is the furthest away from you
a purple one on the top right corner
a purple one on the lower right corner
it's a green one
it's in the middle
right after the two green ones in a row
the closest green one
the green one which is in the middle of the
table
the purple one on the left side
the green one on the top right corner
the green one which is in the middle
on the left side there are three purple ones
the lowest of the three purple ones
the next one up
…
```

Figure 4. Sample N Gram Test File (first 20 lines)

Results

To obtain preliminary results, changes were made to Sphinx-4 Hello World demo files[1] for the sake of time. A short grammar file was successfully used to test a few frequently used words with Snackbot. These words include: *yes, no, orange, cookie, mystery snack, snickers, candy,* and a few others. Some of the words, such as snickers, were not already a part of the dictionary. Therefore, changes were made to the dictionary to include these words with the proper phoneme using the Hidden Markov Model. With this grammar, the application was ran in Eclipse (see Figure 5) and was able to recall sentences that were spoken, though not at a 100% accuracy. At this point, though progress was made, Snackbot was not able to respond orally due to the focus on recognition and less on incorporating the speech mechanism files and coding.

[1] http://cmusphinx.sourceforge.net/wiki/tutorialsphinx4

Discussion and Future Work

The ideal method to test the recognition of a language model is to use the Hello N-Gram or the Live Connect demo. The Hello N-Gram demo, unlike the Hello World demo, uses a language mode instead of a grammar file, offering more flexibility among the grammar used. The Live Connect demo is an action tag demo that shows how to use an action tag parser once a speech is recognized. The Live Connect could be used to plug in code allowing objects in the application to be activated and can therefore use a spoken language system to send accurately responses to be spoken by Snackbot, using the ECMA Script logic.

The results of the Hello Word demo, using the most common words spoken to Snackbot allowed us to get a feel of how Sphinx works and what files to manipulate, reaping desirable results. The accuracy of the test done was dependent on the pace, the volume, and how clearly words were spoken. The more enunciated the words were, using a standard pace of speaking, the more capable the application was able to recall words. Further research would allow more changes to be made to the class file, allowing the voice generator used by the Snackbot, to respond orally and using more than one tester to assess the recognition of words.

Acknowledgement

This research was supported in part by the National Science Foundation (CNS-1042466) and various corporate partners. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or our corporate partners.

References

CMU. 2011. Sphinx: Open Source Toolkit For Speech Recognition. http://cmusphinx.sourceforge.net/

CMU. n.d. Sphinx Knowledge Base Tool. http://www.speech.cs.cmu.edu/tools/lmtool.html

Images Scientific Instruments Incorporated. n.d. *Build a Speech Recognition Circuit*. Retrieved November 21, 2011, from http://www.imagesco.com/articles/hm2007/SpeechRecognitionTutorial01.html

Jurafsky, D., and Martin, J. 2008. *Speech and Language Processing*. Upper Saddle River, NJ: Pearson Prentice Hall.

Kumar, R. (2008) *Human Computer Interaction*. New Delhi, India: Firewall Media.

Lee, M.K., Forlizzi, J., Rybski, P.E., Crabbe, F., Chung, W., Finkle, J., Glaser, E., and Kiesler, S. 2009. The Snackbot: Documenting the design of a robot for long term human robot interaction. In *Proceedings of HRI 2009*, 7 14.

Li, X. 2005. *Combination and Generation of Parallel Feature Streams for Improved Speech Recognition*, Ph.D. Thesis, ECE Department, CMU, February 2005.