# Augmenting User Interfaces with Adaptive Speech Commands

Peter Gorniak
MIT Media Laboratory
20 Ames Street
Cambridge, MA, 02142, USA
pgorniak@media.mit.edu

Deb Roy
MIT Media Laboratory
20 Ames Street
Cambridge, MA, 02142, USA
dkroy@media.mit.edu

## ABSTRACT

We present a system that augments any unmodified Java application with an adaptive speech interface. The augmented system learns to associate spoken words and utterances with interface actions such as button clicks. Speech learning is constantly active and searches for correlations between what the user says and does. Training the interface is seamlessly integrated with using the interface. As the user performs normal actions, she may optionally verbally describe what she is doing. By using a phoneme recognizer, the interface is able to quickly learn new speech commands. Speech commands are chosen by the user and can be recognized robustly due to accurate phonetic modelling of the user's utterances and the small size of the vocabulary learned for a single application. After only a few examples, speech commands can replace mouse clicks. In effect, selected interface functions migrate from keyboard and mouse to speech. We demonstrate the usefulness of this approach by augmenting jfig, a drawing application, where speech commands save the user from the distraction of having to use a tool palette.

## Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Language Acquisition*; I.2.7 [**Artificial Intelligence**]: Natural Language—*Speech recognition and synthesis*

## General Terms

Human Factors, Experimentation

## Keywords

phoneme recognition, machine learning, user modelling, robust speech interfaces

## 1. INTRODUCTION

Spoken input has great potential value as an augmentation to standard graphical user interfaces. In a drawing application, for example, the user can speak commands that would otherwise require him or her to search tool palettes and menus, pulling attention away from the drawing task at hand. A straightforward way to add speech input to an interface is to specify a vocabulary of commands that are mapped to interface actions. To use such an interface, the user must learn (memorize) the vocabulary and associated actions. As an alternative, we have reversed the roles and put the burden of learning vocabulary and associated actions on the system. By doing so, the user is able to shape the system to understand words of his or her choice.

In this paper, we propose an adaptive speech input augmentation to any standard graphical user interface. With this speech layer added to the interface, the user may continue to use the system as before, controlled solely with keyboard and mouse. However, he or she can also use speech to name interface actions such as button clicks. A phoneme recognizer produces a phonetic representation of the utterance, which is associated with the interface action that occurs closest in time. After a few consistent examples, the user can speak the name he or she chooses for the action instead of using the mouse or keyboard.

We show in this paper that training speech commands in this manner is quick, robust, and can occur while the user continues to use the application in a normal way. By monitoring the event queue of Java applications, our system can be attached to any application written in this language without the need to have access to the source code of the application. In this paper, we use jfig, a vector based drawing program written in Java, as an example. Figure 1 shows a typical jfig screen as well as the feedback panel for our speech augmentation system.

Our approach is inspired by work in computational modelling of infant language acquisition [8]. This approach relies on correlation of the visual and the speech modality. We simplify the speech learning problem to only include full utterances delimited by silence, and replace the visual input with user interface events. Treating the multimodal input learning problem as a problem of acquisition of semantics for speech commands distinguishes this research from other multimodal interface approaches. Other approaches typically presuppose full vocabulary speech and pen gesture recognition, as well as a mapping to an existing semantic encoding. Adaptation in these systems, when attempted at

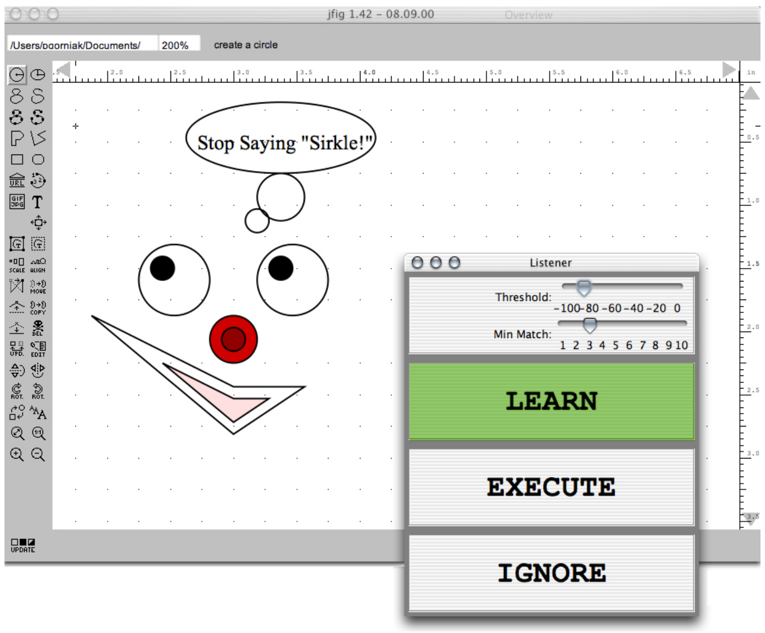

**Figure 1: The jfig drawing application with the added speech learning panel. The panel currently indicates that it accepted the last speech utterance as an example for the jfig tool simultaneously selected by the user.**

all, refers to adapting to the user's input behaviours (e.g. speech timing) or the environment (e.g. noise level) [5]. In contrast, we propose learning the meaning of the user's actions through cross-modal analysis.

## 2. PHONEME RECOGNITION

Spoken utterances are represented as arrays of phoneme[1] probabilities. Acoustic input is first converted into a spectral representation using the Relative Spectral-Perceptual Linear Prediction (RASTA-PLP) algorithm [2]. RASTA-PLP is designed to attenuate non-speech components of an acoustic signal. It does so by suppressing spectral components of the signal which change either faster or slower than speech. First, the critical-band power spectrum is computed and compressed using a logarithmic transform. The time trajectory of each compressed power band is filtered to suppress non-speech components. The resulting filtered signal is expanded using an exponential transformation and each power band is scaled to simulate laws of loudness perception in humans. Finally, a 12-parameter representation[2] of the smoothed spectrum is estimated from a 20ms window of input. The window is moved in time by 10ms increments resulting in a set of 12 RASTA-PLP coefficients estimated at a rate of 100Hz.

A recurrent neural network (RNN) analyses RASTA-PLP coefficients to estimate phoneme and speech/silence probabilities. The RNN has 12 input units, 176 hidden units, and 40 output units. The 176 hidden units are connected through a time delay and concatenated with the RASTA-PLP input coefficients. Thus, the input layer at time $t$ consists of 12 incoming RASTA-PLP coefficients concatenated with the activation values of the hidden units from time $t$*-1*. The time delay units give the network the capacity to remember aspects of old input and combine those representations with fresh data. This capacity for temporal memory has been shown to effectively model coarticulation effects in speech [7]. The RNN was trained off-line using the TIMIT database of phonetically transcribed American English speech recordings [1].

To locate approximate phoneme boundaries, the RNN outputs are treated as state emission probabilities in a Hidden Markov Model (HMM) framework. The Viterbi dynamic programming search [3] [6] is used to obtain the most likely phoneme sequence for a given phoneme probability array. After Viterbi decoding of an utterance, the system obtains (1) a phoneme sequence, the most likely sequence of phonemes which were concatenated to form the utterance and (2) the location of each phoneme boundary for the sequence. Each phoneme boundary serves as a speech segment start or end point. Any subsequence within an utterance terminated at phoneme boundaries is used to form word hypotheses. Additionally, any word candidate is required to contain at least one vowel. This constraint prevents the model from hypothesizing consonant clusters as word candidates. Instead, each candidate is guaranteed to consist of one or more syllables consisting of a vowel and consonant or consonant cluster on either side of the vowel. We refer to a segment containing at least one vowel as a *legal segment*.

A distance metric, $d_A()$, measures the similarity between two speech segments. It is possible to treat the phoneme sequence of each speech segment as a string and use string comparison techniques. This method has been applied to the problem of finding recurrent speech segments in continuous speech [10]. A limitation of this method is that it relies on

[1]We use the set of 40 English phonemes defined in [3].
[2]An all-pole model of the spectrum is estimated using linear predictive coding [4].

[3]The Viterbi algorithm is commonly used in speech recognition applications to efficiently find the most likely HMM state sequence corresponding to an observed observation sequence.

only the single most likely phoneme sequence. A sequence of RNN output contains additional information which specifies the probability of all phonemes at each time instance. To make use of this additional information, we developed the following distance metric.

Let $Q = \{q_1, q_2, \ldots, q_N\}$ be a sequence of $N$ phonemes observed in a speech segment. This sequence may be used to generate a HMM model $\lambda$ by assigning an HMM state for each phoneme in $Q$ and connecting each state in a strict left-to-right configuration. State transition probabilities are inherited from a context-independent set of phoneme models trained from the TIMIT training set. Consider two speech segments, $\alpha_i$ and $\alpha_j$ with phoneme sequences $Q_i$ and $Q_j$. From these sequences, we can generate HMMs $\lambda_i$ and $\lambda_j$. We wish to test the hypothesis that $\lambda_i$ generated $\alpha_j$ (and vice versa).

The Forward algorithm[4] [6] can be used to compute $P(Q_i|\lambda_j)$ and $P(Q_j|\lambda_i)$, the likelihood that the HMM derived from speech segment $\alpha_i$ generated speech segment $\alpha_j$ and vice versa. These likelihoods are not an effective measure for our purposes since they represent the joint probability of a phoneme sequence and a given speech segment. An improvement is to use a likelihood ratio test to generate a confidence metric. In this method, each likelihood estimate is scaled by the likelihood of a default alternate hypothesis, $\lambda^A$:

$$L(Q, \lambda, \lambda^A) = \frac{P(Q|\lambda)}{P(Q|\lambda^A)} \tag{1}$$

The alternative hypothesis is that the HMM was derived from the speech sequence itself, i.e., $\lambda_i^A = \lambda_j$ and $\lambda_j^A = \lambda_i$. The symmetric distance between two speech segments is defined in terms of logarithms of these scaled likelihoods:

$$d_A(\alpha_i, \alpha_j) = -\frac{1}{2}\left\{\log\left[\frac{P(Q_i|\lambda_j)}{P(Q_i|\lambda_i)}\right] + \left[\frac{P(Q_j|\lambda_i)}{P(Q_j|\lambda_j)}\right]\right\} \tag{2}$$

The speech distance metric defined by Equation 2 measures the similarity of phonetic structure between two speech sounds. The measure is the product of two terms: the probability that the HMM extracted from observation A produced observation B, and vice versa. Empirically, this metric was found to return small values for words which humans would judge as phonetically similar.

## 3. INTERFACE AUGMENTATION

Java's reflective capabilities and dynamic loading strategy make the language an excellent candidate for an application independent approach [9]. Java's dynamic loading of classes rids the developer of needing to link with or even know about classes that will be present at runtime. Our system runs as a wrapper to a Java application. Before it starts the application, it hooks itself into the application's event queue and thus sees all event activity within the Java Abstract Window Toolkit and components derived from it. It intercepts each such event that it considers an action (such as a button being pressed or a window closed), submitting it for processing to the speech learning layer before passing it on to the application.

[4]The forward algorithm efficiently computes the probability that an observation sequence was generated by a specific HMM.

When the start of an utterance occurs within a short time window (2 seconds) around an interface event, our system stores the HMM corresponding to the utterance as a candidate speech command for the event. When an utterance occurs in isolation, the system searches through the interface events that have stored utterances associated with them. It computes a score according to Equation 2 for the new utterance and each stored utterance. If the score exceeds a threshold, the utterance pairing is considered a match. Let us denote the number of matches for interface event $e$ $m_e$. The system then computes

$$2\max_e(m_e) - \sum_e m_e \tag{3}$$

in effect weighing the support for the event with the most matches against the support for other events. If this number exceeds the minimum number of matches (2, in the examples given below), the system executes the action with the greatest support, $\arg\max_e(m_e)$.

## 4. EXPERIMENTS AND RESULTS

To evaluate the ease and robustness with which speech commands can be trained, we ran two small experiments in which participants trained the jfig drawing program to obey a set of speech commands equivalent to the tools in the jfig tool palette. In the first experiment, we took a laptop running the application to 5 participants' work desks. Participants wore a head-worn microphone. Each person was given an explanation of how to train the application ("Click on a tool and say what you want to call it"), and was then asked to train five tools of his or her choice five times each. Participants were also told that only examples in which the "LEARN" sign turned green counted as training instances. Participants received feedback after the first two training instances in the form of corrections to the training procedure (participants occasionally thought they should keep the mouse button pressed during their utterance, or they could call the tool by different names in each example). After 25 training examples, participants were asked to speak the name of each tool they had trained. We recorded whether our system selected the correct tool in response to each command. As participants were unfamiliar with the drawing application, they selected names ranging from "circle" for the circle tool to "p-symbol" for the polygon tool and "figure eight" for the closed spline tool. Both in this and in the next experiment, participants frequently spoke during the experiment other than when they were training tools (they spoke to the experimenter, to themselves, and performed other acoustics acts like clearing their throat or laughing). All of these other speech and speech-like utterances resulted in an "IGNORE" response by the system. Table 1 shows the response accuracy across tools in the order tools were trained. The system never selected the wrong tool. Rather, its errors resulted in an "IGNORE" response to a valid speech command. We observed that after the first few training instances and receiving feedback from the

| | Tool 1 | Tool 2 | Tool 3 | Tool 4 | Tool 5 |
|---|---|---|---|---|---|
| Accuracy | 60% | 100% | 100% | 100% | 100% |

**Table 1: Results for five participants training five tools each. Tools are presented in the order trained.**

experimenter subjects quickly became comfortable with the training procedure. The lower accuracy for the first tool can thus be attributed to this adjustment period, in which some inconsistent speech utterances were recorded for the tool. To show that this adjustment period does not pose a problem in the long run, we subsequently asked subjects with lower accuracy for the first tool to provide another 5 training instances for this tool. After this additional training, recognition accuracy for the first tool was 100%.

| | Participant 1 | | Participant 2 | |
|---|---|---|---|---|
| | consistent | overall | consistent | overall |
| Accuracy | 56% | 84% | 70% | 84% |

**Table 2: Results for two participants training 25 tools each.**

To test whether the training procedure scales to a more significant number of commands, we asked another two participants to train speech commands for 25 tools. Table 2 shows the response accuracy of the system for each of the two participants. We divide the performance into *consistent* for the commands that are recognized consistently across several repetitions and *overall* for commands that are recognized only sporadically. The lower accuracy scores show that performance deteriorates when many commands are trained. This is mainly due to the fact that the phoneme recognizer does not consistently capture unambiguous phoneme traces for short utterances with few vowels. For example, words like “arc”, “gif” and “link” produce low confidence matches that lead to rejections. Another source of mistakes are phonetically similar utterances, for example “align” and “line”. Phonetically distinct utterances (“circle”, “clockwise”, “out”) and long utterances (“rounded rectangle”, “open b-spline”) are easily distinguished and lead to almost no mistakes. Performance can be improved by selectively providing more examples for tools that are not recognized, or training a different utterance for those tools.

## 5. SUMMARY AND FUTURE WORK

We have presented a simple, yet easy-to-use speech augmentation system for unmodified applications. By monitoring the event queue of Java applications and correlating events with speech utterances as detected by a recurrent neural network phoneme recognizer, the system quickly learns speech commands for events in the graphical user interface of the monitored application. These speech commands can be used to drive the application via speech instead of mouse and keyboard. Our system has the advantages that training occurs during normal application use, that speech recognition is robust due to the small and user-specific vocabulary, and that it does not require the user to learn unfamiliar commands. The results of two small studies show that training is indeed easy and performance is robust for small sets of commands. For larger sets performance deteriorates due insufficient distinguishability of speech commands. However, users can rectify such problems by further training and choosing more distinct command utterances. Furthermore, even with only a few trained speech commands, the system provides useful added convenience, especially for functions such as “zoom in” or “delete” which would otherwise distract the user’s attention from the task at hand.

Training for larger sets of utterances would be made easier by better feedback to the user. For example, the system could warn the user if two sets of utterances for different tools are acoustically confuseable. The system could also silently prune outlier utterances for a tool if they are not similar to the bulk of utterances for that tool.

In the future, we plan to expand the system to take into account the actual content of the drawing or design surface in an application, instead of paying attention only to standard user interface controls. Doing so will require in-depth models of how the meanings of words and phrases are grounded in the visual content of the application program.

## 6. ACKNOWLEDGMENTS

The authors would like to thank Dr. Norman Hendrich for generously agreeing to let us use his jfig drawing program for our research.

## 7. REFERENCES

[1] J. Garofolo. *Getting Started with the DARPA TIMIT CD-ROM: An Acoustic Phonetic Continuous Speech Database*. National Institute of Standards and Technology (NIST), Gaithersburgh, MD, 1988.

[2] H. Hermansky and N. Morgan. Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing*, 2(4):578–589, October 1994.

[3] K. Lee. *Large-vocabulary speaker-independent continuous speech recognition: The SPHINX system.* PhD thesis, Computer Science Department, Carnegie Mellon University, 1988.

[4] A. Oppenheim and R. Schafer. *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[5] S. Oviatt, P. Cohen, L. Wu, J. Vergo, L. Duncan, B. Suhm, J. Bers, T. Holzman, T. Winograd, J. Landay, J. Larson, and D. Ferro. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions. *Human Computer Interaction*, 15(4):263–322, August 2000.

[6] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

[7] T. Robinson. An application of recurrent nets to phone probability estimation. *IEEE Trans. Neural Networks*, 5(3), 1994.

[8] D. Roy and A. Pentland. Learning words from sights and sounds: A computational model. *Cognitive Science*, 26(1):113–146, 2002.

[9] Sun Microsystems. *Java Development Kit (http://java.sun.com)*, 2003.

[10] J. Wright, M. Carey, and E. Parris. Statistical models for topic identification using phoneme substrings. In *Proceedings of ICASSP*, pages 307–310, 1996.