SpeechMap: An API for Speaker-Dependent, Language-Independent Voice Interfaces

Abdul Hannan Kanji, Swathi Veeradhi, Mayank B A
Department of Computer Science and Engineering
PES Institute of Technology
Bangalore, India

Viraj Kumar
Department of Computer Science and Engineering
PES University
Bangalore, India
viraj.kumar@pes.edu

ABSTRACT
Voice interfaces are a promising way to make software applications (apps) accessible to users with visual or motor impairments. Language-dependent speech recognition algorithms can achieve good accuracy on mobile devices with limited computing power, and they are speaker-independent with one important caveat: they assume that users do not have speech disorders. These algorithms have been cloaked beneath APIs that make it easy for developers to create apps with voice interfaces, while designers continue to improve the underlying algorithms. In this paper, we propose an API for speaker-dependent, language-independent voice interfaces. We argue that such an API eases the task of creating accessible apps for at least two categories of users: (1) those who have certain kinds of speech disorders, and (2) those who lack fluency in languages for which speech-recognition algorithms are currently available. We also illustrate the potential of this API to enhance existing apps and create new types of apps.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Input devices and strategies*; K.4.2 [**Computers and Society**]: Social Issues – *Assistive technologies for persons with disabilities*

General Terms
Design, Human Factors, Standardization.

Keywords
Human-computer interaction, voice interfaces, mobile apps.

1. INTRODUCTION
The proliferation of low-cost mobile computing platforms has led to the creation of software applications (apps) that interact with unprecedented numbers of users. These apps typically present users with visual information, and they expect users to interact with the app by performing gestures such tapping and swiping the screen, or clicking buttons. Assistance is available for users who are blind (or otherwise unable to read small displays) in the form of pre-installed screen-readers such as Android's TalkBack [1] and iOS's VoiceOver [2], but these are effective for apps with largely textual interfaces, and for users who can understand one of the supported languages (Android supports 13 languages at present [3], for instance). Input devices such as Braille keyboards are available, and there is active research in designing audio-tactile keypads [7]. For users with motor impairment, assistance is once again available in software (in the form of larger icons or alternative gestures) and hardware (in the form of accessory controllers). Even so, studies indicate that users with variable motor abilities (especially hand tremors) find it difficult to interact with apps on mobile devices [4] and PCs [5]. Furthermore, the benefits of additional hardware are often offset by the additional costs and bulk they impose [8].

Conference'10, Month 1–2, 2010, City, State, Country.

Voice-interfaces permit users to interact with apps by uttering short commands, and their only hardware requirement (a microphone) is part of the standard equipment on most mobile devices. They have proved to be popular with blind users [6], the elderly [9], and are being adopted by users with motor-impairments (e.g., [10]). Voice-based inputs can be inaccurate (particularly in noisy environments [11]), and their use raises etiquette and privacy concerns [8], but their ability to simplify app-interactions for many users (including those without visual or motor impairments) is well-recognized, and has spurred research in algorithms for automatic speech recognition (ARS). Developers do need not know the technical nuances behind these advances, and application program interfaces (APIs) such as Android's SpeechRecognizer [12] and iOS's OpenEars [13] have been developed so that apps can invoke speech recognizers as per their need, oblivious to the (increasingly better) algorithms being executed internally. Given this simplicity of design, we expect to see an increasing number of apps requiring users to interact orally. Such apps are likely to remain inaccessible for at least two groups of users: (1) those with speech disorders (algorithms for speech-recognition are rarely tuned for such inputs [5]), and (2) those who lack fluency in any of the languages supported by these algorithms. Although speakers whose native language is not supported often learn at least one of the supported languages as part of their formal education, we note that people with disabilities are likely to constitute a larger fraction of group (2) than their demographic proportion. This is because there are well-documented differences in access to quality education and learning outcomes between students with and without disabilities (e.g., [14], [15]).

In this paper, we propose an alternate API that relies on algorithms that detect speech patterns (but do *not* attempt to recognize words), and can cater to the requirements of at least some of these users. For users with speech disorders, our approach requires utterances to be reasonably *consistent*: mispronunciations can be recognized as long as they are pronounced similarly each time. This includes users with phonetic and phonemic disorders, voice disorders, dysprosody, dysarthria, specific language impairment (SLI), and (in certain cases) aphasia. Our approach is unlikely to aid users

whose speech disorders lead to inconsistent vocalizations of words, such as cluttering, stuttering and apraxia. The strategy we propose should also aid those who can speak at least one language fluently, although this remains to be fully tested. Our initial experiments, conducted on a small sample of Indian languages, are promising for a country where the potential for voice-based interfaces to provide information services to millions of illiterate people has been demonstrated [17].

The rest of the paper is organized as follows. Section 2 details previous work that addresses similar concerns, and explores their limitations. We present our SpeechMap API in Section 3, and we contrast it with Android's SpeechRecognizer [12] to highlight the differences in purpose of the two APIs. In Section 4, we illustrate the use of our API with two case studies: the game VBHangman [16] and one of our own apps. Finally, we present our conclusions in Section 5.

2. RELATED WORK

Speech-recognition algorithms typically require users to calibrate parameters by uttering pre-determined phrases. Users who cannot speak the associated languages and those with speech disorders find it difficult or impossible to use such tools [5]. Two voice-based alternatives have been proposed. In the first approach, known as Non-verbal Vocal Input (NVVI), users associate whistles and hums with specific commands during a training phase, after which the software executes the desired commands by interpreting the user's vocal signals [18]. This approach is effective for users with speech disorders that severely limit their vocal range, but it can be cumbersome for those who can produce a rich variety of sounds, and is error-prone [5]. The second approach modifies the standard speech-recognition process-flow (see Fig. 1) by intercepting the recognizer at the point where it detects phonemes, but before it interprets these as words (i.e., the output of the *Acoustic Models*, before they enter the *Decoder*). In one such study [19], commands issued by users in Gujarati were converted into US English phonemes. Despite the fact that there is no English equivalent for certain Gujarati phonemes, the software adequately recognized a simple vocabulary of commands. Our approach is inspired by these two alternatives. For our purpose, we believe it is better to truncate the speech-recognition process (Fig. 1) even earlier, after the *Signal Processing* stage where the output is merely a sequence of vectors time-stamped relative to the start of speech. We believe that these vectors encode sufficient information to distinguish commands in a language-agnostic manner, and our implementation of the SpeechMap API (Section 3) makes this assumption. If this assumption is found to be invalid, or if a better encoding of information is found, we still believe that our API will remain a useful contribution.

3. THE SPEECHMAP API

In order to relieve app developers from the burden of understanding the details of speech-recognition algorithms, an easy-to-use API is often made available. Android's official speech recognition API, the somewhat prosaically named SpeechRecognizer [12], permits app developers to write code in response to words that have been recognized. The decoded words are presented as a sequence[1], and

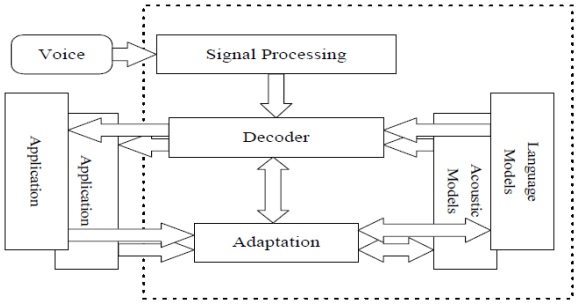

Figure 1. Basic architecture of a speech recognizer [20].

an associated sequence of "confidence scores" is also available[2]. Such an API is unsuitable for our purpose, because we do not wish to interpret the user's vocalizations as words. Our SpeechMap API is loosely based on Java's **`Map`** interface [21], and permits app developers to associate **`Speech`** objects with objects of any other type. In our implementation, **`Speech`** objects encode sequences of numerical vectors[3] as explained in Section 2, but any representation of the vocalization that supports a notion of *similarity* will suffice for our API[4].

The core functionality of our API is given in Table 1. There are two methods for associating speech with a target object (which can refer to anything that the developer demands). The first *put* method simply associates a given **target** with **speech**, replacing any existing association that **speech** may have had. The second *put* method is more interesting: it prohibits associating **speech** *s* with **target** *t* if there is *already* some other speech object s_1 such that is similar to *s* and associated with a target. This functionality allows the app developer to warn the user that the app may fail to distinguish the voice samples that give rise to *s* and s_1. In such a situation, the app user would be expected to provide an alternate command.

Table 1. Core functionality of the SpeechMap API

Method	Description
put(**speech**, **target**)	Unconditionally adds a new (**speech**, **target**) pair, and returns the target (if any) previously associated with **speech**
put(**speech**, **target**, **sim**)	Adds new (**speech**, **target**) pair and returns null if no existing speech is *similar* (similarity more than **sim**) to **speech**, else does not add anything but returns all targets whose speech is similar to **speech**
get(**speech**, **similarity[ ]**)	Returns an array whose length is equal to that of the **similarity[ ]** array, containing the best targets for **speech**, ordered in decreasing order of similarity, and fills the **similarity[ ]** array accordingly
get(**speech**, **sim**)	Returns all targets whose associated speech's similarity with **speech** is at least **sim**, in decreasing order of similarity

[1] Technically, this sequence is an **`ArrayList<String>`** object.

[2] The Android documentation specifies that an "array of confidence values *might* also be given" (emphasis added).

[3] Specifically, the MFCC vector, the 1st-order delta MFCC, and the 2nd-order delta MFCC, each of 13 dimensions, resulting in a 39-dimensional sequence of vectors [22].

[4] We extend the standard notion of Euclidean distance between vectors to sequences of vectors.

Our API also provides two *get*() methods for retrieving targets associated with a given **speech**. The app developer chooses the first alternative when she wishes to retrieve the *k* best matches to the given **speech**, where *k* is a parameter of her choice. This method orders the returned targets in decreasing order of similarity (whose numerical value is also available). The app developer can therefore arrange these matches for the user to choose from (with the best matches easiest to select), and discard any targets whose similarity is below a threshold of her choice. The second *get*() method allows the app developer to specify a threshold **sim** and retrieve all targets whose similarity with the given **speech** is at least **sim**.

4. CASE STUDIES

In this section, we sketch two case studies (both of which we are presently evaluating) to illustrate the usage of our API. First, we explain how an existing app for blind children can be modified to enhance its accessibility to blind children with motor impairment. Next, we consider an app that permits a blind fruit vendor to record transaction items in his native language (Kannada), compute the total owed by the customer, and speak this value in English one numeral at a time (which he can understand).

4.1 VBHangman with SpeechMap

VBHangman illustrates the power of the VBraille interface to create engaging educational games for blind children [16]. The study reported some participants facing "difficulty with the two-finger swipe, indicating that multi-touch gestures may not be suitable for children." The game asks players to complete a word where some of the letters are missing, where instructions and feedback are spoken in English. To use our API, VBHangman would be modified to include a training phase where the app spoke out each letter of the alphabet and ask the player to repeat it. The **target** associated with **speech** would be the individual letters. Thereafter, the app would proceed with the existing game. When it comes to asking the player to input a letter, the app would record the voice sample, and use the first version of the *get*() method to identify (say) the 3 best matches. Instead of asserting "You entered the letter …", the modified app would ask "Did you mean (*letter corresponding to the best match*)?", and after a pause could proceed further down the list: "or did you mean (*next letter*)?".

4.2 Fruit Vendor app with SpeechMap

We have developed an app to assist a blind fruit-seller to compute the total amount his customers owe by speaking out individual items. The training phase of this app is more complex, because the fruit-seller offers a variety of discounted rates for items depending on the customer (friends and relatives get a discount) and the time of day (any unsold items at the end of the day represent a total loss, so he lowers prices as the day proceeds). Thus, in addition to the list of fruits, the vocabulary must include keywords representing the various types of discounts. Since the fruit-seller works on a noisy street, the first version of the *get*() method often returns poor matches. As a result, the feedback we received for the first version of our app was poor. We have subsequently modified the app to use the second version of *get*() with a sufficiently high threshold. If no match is found, the app prompts the fruit-seller to repeat himself.

5. CONCLUSIONS

The applications of the API we have reported in Section 4 are works in progress, but the purpose of this paper is primarily to argue for the necessity of a new kind of API that permits app developers to leverage advances in signal and speech processing, in order to build apps that use voice interfaces that are language-independent (even if they are speaker-dependent). The dominant trend in speech-recognition research is towards improving speaker-independent algorithms for widely spoken languages. Apps developed to take advantage of these capabilities will add another dimension of inaccessibility for the kinds of users whose needs we have attempted to address in this paper. On the other hand, if our API is widely adopted, we believe it will spur the development of more sophisticated signal processing techniques to improve the accuracy of apps built using this API, which can enhance their accessibility.

6. REFERENCES

[1] Android. 2015. Accessibility. http://developer.android.com/design/patterns/accessibility.html Accessed May 7, 2015.

[2] Apple. 2015. iOS: A wide range of features for a wide range of needs. https://www.apple.com/accessibility/ios/ Accessed May 7, 2015.

[3] Google. 2015. Google Text-to-Speech. https://play.google.com/store/apps/details?id=com.google.android.tts&hl=en

[4] Kyle Montague, Hugo Nicolau, and Vicki L. Hanson. 2014. Motor-impaired touchscreen interactions in the wild. In Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility (ASSETS '14). ACM, New York, NY, USA, 123-130. DOI= http://doi.acm.org/10.1145/2661334.2661362.

[5] Torsten Felzer and Stephan Rinderknecht. 2013. How someone with a neuromuscular disease experiences operating a PC (and how to successfully counteract that). In Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13). ACM, New York, NY, USA, Article 29.

[6] Azenkot, S. and Lee, N.B. Exploring the Use of Speech Input by Blind People on Mobile Devices. In Proc. 15th International ACM SIGACCESS Conference on Computers and Accessibility, ACM (2013), 11:11-11:18.

[7] Maria Claudia Buzzi, Marina Buzzi, Barbara Leporini, and Amaury Trujillo. 2014. Designing a text entry multimodal keypad for blind users of touchscreen mobile phones. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility* (ASSETS '14). ACM, New York, NY, USA, 131-136.

[8] Kane, S.K., Jayant, C., Wobbrock, J.O., and Ladner, R.E. Freedom to roam: a study of mobile device adoption and accessibility for people with visual and motor disabilities. In Proc. 11th international ACM SIGACCESS, (2009).

[9] François Portet, Michel Vacher, Caroline Golanski, Camille Roux, Brigitte Meillon. Design and evaluation of a smart home voice interface for the elderly – Acceptability and objection aspects. Personal and Ubiquitous Computing, Springer Verlag (Germany), 2013, 17 (1), 127-144.

[10] Xiomara Figueroa Fontánez and Patricia Ordóñez. 2014. Improving programming interfaces for people with limited mobility using voice recognition. In Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility (ASSETS '14). ACM, New York, NY, USA, 331-332.

[11] Leporini, B., Buzzi, M.C., and Buzzi, M. Interacting with mobile devices via VoiceOver: usability and accessibility issues. In Proc. 24th Australian Computer-Human Interaction Conference, (2012), 339-348.

[12] SpeechRecognizer. 2015. SpeechRecognizer class overview. http://developer.android.com/reference/android/speech/SpeechRecognizer.html Accessed May 7, 2015.

[13] OpenEars. 2015. OpenEars – iPhone Voice Recognition and Text-To-Speech. http://www.politepix.com/openears Accessed May 7, 2015.

[14] J. D. Gabrieli. 2009. Dyslexia: a new synergy between education and cognitive neuroscience. Science, 325(5938), 280-283.

[15] Blackburn, C. M., Spencer, N. J., and Read, J. M. 2010. Prevalence of childhood disability and the characteristics and circumstances of disabled children in the UK: secondary analysis of the Family Resources Survey. BMC pediatrics 10 (1), 21.

[16] Lauren R. Milne, Cynthia L. Bennett, Richard E. Ladner, and Shiri Azenkot. 2014. BraillePlay: educational smartphone games for blind children. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility* (ASSETS '14). ACM, New York, NY, USA, 137-144.

[17] Patel, N., Agarwal, S., Rajput, N., Kumar, A., Nanavati, A., Dave P. and Parikh, T.S. 2008. Experiences Designing a Voice Interface for Rural India, Spoken Language Technology Workshop (SLT'08), 21-24.

[18] Watts, R. and Robinson, P. Controlling computers by whistling. In Proceedings of Eurographics UK, 1999.

[19] J. Sherwani, N. Ali, S. Mirza, A. Fatma, Y. Memon, M. Karim, R. Tongia, and R. Rosenfeld. 2007. Healthline: Speech-based Access to Health Information by Low-literate Users, International Conference on Information and Communication Technologies and Development (ICTD'07).

[20] Huang, X., et al., From Sphinx-II to Whisper - Make Speech Recognition Usable, in Automatic Speech and Speaker Recognition, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, Klewer Academic Publishers.

[21] Map (Java Platform SE 7). 2015. http://docs.oracle.com/javase/7/docs/api/java/util/Map.html Accessed May 7, 2015.

[22] Huang, X., Acero, A. and Hon, H. 2001. Spoken language processing: a guide to theory, algorithm, and system development, Prentice Hall, NJ.