SIGNAL PROCESSING TOOLS FOR SPEECH RECOGNITION[1]

Hualin Gao, Richard Duncan, Julie A. Baca, Joseph Picone

Institute for Signal and Information Processing, Mississippi State University
box 9571, Mississippi State, Mississippi 39762
phone: +1 662 325 8335, email: {gao, duncan, baca, picone}@isip.msstate.edu
web: www.isip.msstate.edu

Abstract

This paper describes the design and development of a set of signal processing software tools for speech recognition. The tools were developed for inclusion in a comprehensive public domain speech recognition toolkit. We describe the design philosophy underlying the development of the tools as well as the key features that enable realization of our design goals of modularity, extensibility, and usability. A GUI-based configuration tool is presented that allows complicated, multi-pass front end algorithms to be created using a graphical editor and a library of fundamental algorithm components. We also discuss results of tests to verify the correctness and usability of the tool set, including benchmarks on SWITCHBOARD, WSJ0 and the Aurora Large Vocabulary tasks.

1. Introduction

The Institute for Signal and Information Processing (ISIP) at Mississippi State University provides a comprehensive public domain toolkit [[1]] for performing large vocabulary speech recognition research. Several differentiating features of this toolkit are its ease of use, extensibility, and educational components. In this paper we describe the design and implementation of its signal processing components, which represent an excellent example of how we have provided a powerful GUI-based environment to perform signal processing research and education.

An overview of a speech recognition system is shown in Figure 1. The tool described in this paper deals with the block known as the Acoustic Front End which encapsulates most of the signal processing portions of a speech recognition system. Signal processing tools extract feature vectors from speech data, and thus play a critical role in the development of speech recognition systems. Also referred to as front end tools, many signal processing toolkits are currently available. MATLAB is one example of the most popular commercial products [[2]]. Such toolkits provide powerful computation and analysis capabilities, and sophisticated graphical interfaces. Nonetheless, they also contain serious deficiencies that limit their usefulness in a research environment.

The first deficiency concerns the need for programming when researchers wish to evaluate new ideas using existing algorithms. Second, adding new algorithms to such toolkits requires modifying the base code of the existing system, a potentially time-consuming and costly undertaking that can significantly impede many research efforts. Third, special problems for speech recognition front ends exist, such as synchronization and buffering of data along the data flow graph. Of even greater importance and difficulty, data preparation for algorithms that require multiple frames of data, such as windows and differentiation, can be problematic.

To address these issues and to provide users with a powerful signal processing tool that requires no programming, we have developed a modular, flexible environment for signal processing. The key differentiating characteristics of our system include[[1]]:

- Competitive technology with maximum flexibility;
- Well-documented APIs to facilitate new programming;
- An object-oriented software design in C++;

This paper first presents our software design rationale and approach for achieving maximum system modularity and usability. It then describes the details of an environment of GUI-based tools we developed following this rationale. This environment enables users to implement front ends by drawing block diagrams of signal processing functions without any programming. Finally, we present results of experiments conducted to test and verify the correctness of the tools.

2. Signal Processing Software Design

Research in the area of speech recognition requires the development of large applications in a relatively short period of

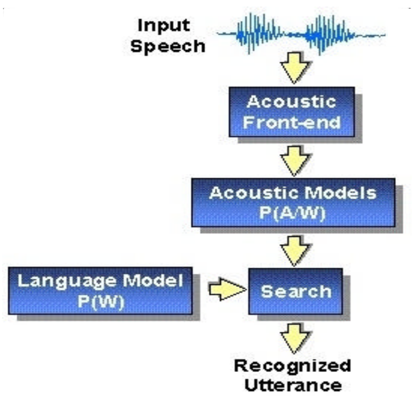

Figure 1: An overview of a speech recognition system.

1. This material is based upon work supported by the National Science Foundation under Grant No. EIA-9809300. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

time. Unfortunately, however, many ideas remain unexplored due to the effort such development requires, including rewriting common functions or debugging low-level issues such as file I/O. To address these needs, we designed a large, hierarchical software environment to support advanced research in all areas of speech recognition, including signal processing. This environment contains ISIP foundation classes (IFCs) that provide features ranging from complex data structures to an abstract file I/O interface. IFCs are implemented as a set of C++ classes, organized as libraries in a hierarchical structure. They are targeted for the needs of rapid prototyping and lightweight programming without sacrificing efficiency. Some key features include:

- Unicode support for multilingual applications;
- memory management and tracking;
- system and I/O libraries that abstract users from details of the operating system;
- math classes that provide basic linear algebra and efficient matrix manipulations;
- data structures that include generic implementations of hash tables, lists, and other things essential for developing flexible and simple speech recognition code.

The IFCs provide support for users to develop new approaches without rewriting common functions. The software interfaces are carefully designed to be generic and extensible. This design approach facilitates overall system development that is modular and flexible, yielding a high level of usability.

We developed our signal processing toolkit to stringently adhere to the IFC design philosophy and framework. The software described in this paper involves primarily the Algorithm library [[3]] in the IFC class hierarchy. At the outset, it was clear that the tools must not only allow a wide selection of algorithms, but also the ability to vary every parameter of each algorithm easily and finally, provide users an efficient environment for evaluating new research ideas. Thus, the design requirements for these tools included:

- a library of standard algorithms to provide basic digital signal processing (DSP) functions;
- an ability to easily add new algorithm classes and functions without modifying existing algorithm classes;
- a block diagram approach to describing algorithms to realize rapid prototyping without programming.

Fulfilling the first requirement enabled users to directly realize a single algorithm such as a window with simple programming by building an algorithm object, and calling its functions. Meeting the second requirement allowed users to enhance system capabilities according to new requirements and to extend to new DSP areas. We designed and implemented an algorithm library and a signal processing library. The object-oriented design of the algorithm library, using inheritance with an abstract base class, greatly enhanced the extensibility of the software. Further details of the interface contract are given in Section 2.1.

To meet the third requirement, we developed a signal processing control tool and a signal processing configuration tool. The procedure by which users employ the tools and libraries can be described as follows: First, the signal processing configuration tool is used to graphically specify the sequence of algorithms and their configuration using a block diagram. This is saved to a file containing a description of the block diagram. This description uses a graph data structure containing components, each of which has its own configuration. Second, a control tool accepts the speech data file and the recipe files produced in the first step as input. It then parses the recipe file using functions provided by the signal processing library to obtain the necessary information for each algorithm. Finally, it applies the corresponding algorithm functions to process the input speech data by calling the correct method in the algorithm library. Because the algorithm library is fundamental to all other libraries and tools, it is described first in the following section.

2.1. Algorithm Library

The algorithm library contains a collection of signal processing and support algorithms implemented as a hierarchy of C++ classes. The implementation of this hierarchy using an abstract base class, AlgorithmBase, and virtual functions or methods that comprise the interface contract, is perhaps the single most important feature, since it makes the library easily extensible. All algorithm classes are derived from this base class. AlgorithmBase class also defines the interface contract, specifying virtual functions that all Algorithm classes must provide, and centralizes useful protected data common to all algorithms, such as sample frequency and frame duration. The interface contract is summarized in Table 1:

```
Processing:
   virtual boolean init();
   virtual boolean apply();

Configuration:
   virtual const String& className() const;
   virtual long getLeadingPad() const;
   virtual long getTrailingPad() const;
   virtual float getOutputSampleFrequency() const;

Debugging:
   boolean displayStart();
   boolean displayFinish();
   boolean displayChannel();
   Boolean display();
```

Table 1: An overview of the interface contract for the Algorithm classes.

Note that the key computational steps of any algorithm are limited to two functions: *init* and *apply*. The remaining functions simply facilitate configuration and debugging. The configuration functions deal with retrieving information from the specific algorithm that is needed to coordinate I/O processing. For example, *getLeadingPad* and *getTrailingPad* are used to determine the amount of delay a specific algorithm introduces so buffers can be adjusted accordingly. The debugging methods were introduced to allow users to setup debugging flag within the signal flow graph, and to see intermediate output.

Expanding the collection of algorithms supported in our Algorithm library is the subject of on-going research. Our current offerings can be sorting into two categories: basic DSP and support. The basic DSP components include commonly used algorithms. The support components allow high-level manipulation of data flow through block diagrams. Together, they provide a unique and powerful set of signal processing capabilities, some of which include: multi-pass processing of a signal; automatic handling of arbitrary amounts of prior and future data when a recipe is created; processing of a signal, saving a constant derived from that signal to a file, and reloading the constant.

The basic DSP components include most commonly used algorithms, such as correlation (e.g., autocorrelation and covariance), spectral transformations (e.g., Fourier transform and Cepstrum), and linear prediction. There are also lower-level classes that provide building blocks for more complicated algorithms. These include components such as windows, filters, and energy. Each algorithm contains numerous algorithm and implementation choices, as well as parameter configuration options. Most popular algorithms that have been used in the past 20 years in speech recognition are included in this inventory.

The support components provide primitive debugging tools as well as the ability to manipulate feature streams. Examples include Constant, which provides a mechanism for applying global constants to a signal, and Math, which is used to form weighted linear combinations of functions of vectors. Combined with the basic DSP algorithms, they yield a powerful set of signal processing capabilities.

To summarize, the algorithm library serves as the foundation for all other signal processing tools and libraries. Its object-oriented design and implementation make it extensible and flexible. The signal processing configuration tool allows users easy access to these algorithms and is described in the following section.

2.2. Signal Processing Configuration Tool

We developed a Java GUI tool, shown in Figure 2, to provide users a block diagram approach to designing front ends. We chose the Java language to allow the tool to run across a wide range of platforms, including Microsoft Windows, and to give the tool an industry-standard look and feel. This tool allows users to select algorithms from an inventory of predefined components, and to connect and configure these components using standard graph drawing tools. Each component represents one algorithm for which the user can specify how to process data; each arc represents a data flow from one algorithm to another. To create a block diagram, the user selects the desired algorithm from the component menu, connects each component using directed arcs, configures each component in the diagram, and saves the configuration or "recipe" into a file. The control tool, described in Section 2.3, uses this file to complete the signal transformation process.

The example block diagram shown in Figure 2 illustrates many unique capabilities of the configuration tool. This example accepts one signal input and applies two sequences of algorithms or data flows in parallel to process the signal. The leftmost sequence (data flow) computes a vector of linear prediction coefficients from a window of data extracted from the signal. The rightmost sequence (data flow) applies a filter to the signal and computes energy using a different window of data, producing a single energy value. The results of each data flow are concatenated via the connection (Conn) component to produce a single vector. The statistics (Stat) component then computes an average from this vector, and finally the constant (Cons) component stores this constant to a file.

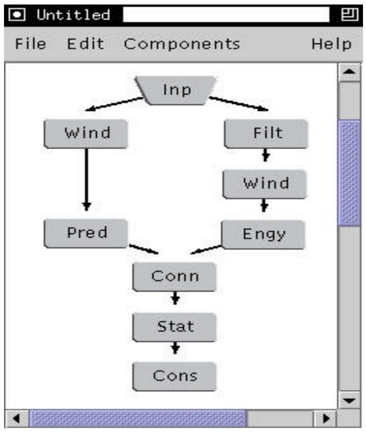

Figure 2: A screenshot of a configuration tool that allows users to create new front ends by drawing signal flow graphs.

This example illustrates several important features of the tool. First, all synchronization and buffering of data between components within a single data flow and across data flows is automatically handled by these tools. The user need only draw the block diagram to indicate how the signal should be processed, without concern for data synchronization or buffering. In addition, the example illustrates the support provided for multi-pass processing. The constant saved to a file can be easily reloaded as input to the same data flow diagram or differing diagrams. All of these capabilities empower researchers to explore ideas freely without the heavy programming burden that might otherwise be incurred.

Finally, to increase the extensibility of the tool, algorithms are presented in the interface through the components menu, populated from a resource file. All algorithms appearing in this menu are read from the resource file. Adding a new algorithm requires simply including a description into the resource file according to its format. No modifications to the source code of the signal processing control tool itself are required.

2.3 Signal Processing Library and Control Tool

The signal processing library is a collection of specially designed modules, implemented as C++ classes, which serve as an interface between the block diagrams, created by the GUI configuration tool, and the computation algorithms, described in Section 2.1. It should be noted that the work of the signal processing library is hidden from the user by default. Its functions include: parsing the file containing the recipe created by the user with the configuration tool; synchronizing different paths along the block flow diagram contained in this file; preparing input/output data buffers for each algorithm, particularly for those requiring multiple frames of data, such as windows or calculus; scheduling the sequences of required signal processing operations; processing data through the flow defined by the recipe; and finally, managing conversational data.

An important attribute of the signal processing control tool concerns its compatibility with other components of our software. The control functions are embedded in the recognizer so that both tools use exactly the same code base. This further enhances the usability of the toolkit, enabling researchers to more easily achieve consistency in experimental results by using the same data in recognition as that used in feature extraction. Also, the fact that the front end is embedded in the recognizer allows live input demos to be easily created, again by supporting use of the same data from feature extraction in recognition.

3. Experimental Results

We tested the quality of our toolkit along two dimensions, correctness and usability. To verify the correctness of the computation results, we have successfully built several complex front ends, including an industry standard front end based on Mel-frequency cepstrum coefficients (MFCCs) [4]]. Our testing procedure entailed first comparing the data generated from the general purpose tools described in this paper to similar data generated from a prior version of our software that has been publicly available for several years, but contained less general implementations of many algorithms. Since there are subtle differences in the way the components are implemented, byte by byte comparisons of the data are not always possible or desirable.

Hence, in addition to directly comparing values in the feature vector streams, we also ran several recognition experiments, including Switchboard (SWB) [[5]] and Wall Street Journal (WSJ) [[6]]. As one example, we compared the correctness of our results against comparable baseline systems used in the Aurora evaluations on the 5K WSJ0 task [[7]]. Front ends created using our general purpose tools matched performance on these tasks achieved using the older versions of the software. For example, we achieved an 8.3% WER using our MFCC front end recipe, and this matches the results reported in [[4]].

Next, we assessed and enhanced the usability of our tools through extensive user testing conducted over the course of many workshops [[7]]. As part of this testing, we administered a user survey derived from the Questionnaire for User Interaction Satisfaction (QUIS), a measurement tool designed for assessing user subjective satisfaction with the human-computer interface [[8]]. Several features of the interface were modified or enhanced as a result of these tests. General examples include reductions in the number of menus, number of menu options, changes in wording of menu options, and modifications to the behavior of the drawing tool itself.

As a more specific example, the original GUI design contained a Configuration menu, containing Input and Output components to be configured, and an Algorithms menu, containing algorithm components. Usability tests indicated users did not find this set of menu choices intuitive; input and output were included with algorithms under one menu, Components. This allowed configuration of all block diagram components.

4. Conclusions

This paper has presented the signal processing component of our public domain speech recognition toolkit. This component was designed and developed in adherence to our philosophy of providing a flexible, extensible software environment for speech recognition researchers. Our goal was to enable researchers to explore ideas freely, unencumbered by low-level programming issues. To achieve this goal, we implemented several critical features in our signal processing software tools, including a library of standard algorithms for basic DSP functions, the ability to add new algorithms to this library easily, and a GUI-based configuration tool for creating block diagrams to describe algorithms, allowing rapid prototyping without programming. We have tested and verified this tool for both correctness and usability. It empowers researchers to easily build state-of-the-art front end systems for speech recognition. We continue to monitor feedback from our user community in order to maintain the highest quality of the tool.

5. References

[1]. K. Huang and J. Picone, "Internet-Accessible Speech Recognition Technology," presented at the IEEE Midwest Symposium on Circuits and Systems, Tulsa, Oklahoma, USA, August 2002 (see http://www.isip.msstate.edu/projects/speech*)*.

[2]. The MathWorks, Inc., Natick, MA, USA (see http://www.matlab.com/).

[3]. R. Duncan, H. Gao, J. Baca and J. Picone, "The Algorithm Classes," ISIP, Mississippi State University, MS State, MS, USA, March 2003 (see http://www.isip.msstate.edu/projects/speech/software/documentation/class/algo/).

[4]. N. Parihar, *et al.*, "Performance Analysis of the Aurora Large Vocabulary Baseline System," *Proc. of Eurospeech '03*, pp. 337-340, Geneva, Switzerland, September 2003.

[5]. R. Sundaram, J. Hamaker, and J. Picone, "TWISTER: The ISIP 2001 Conversational Speech Evaluation System," *Proceedings of the Speech Transcription Workshop*, Linthicum Heights, Maryland, USA, May 2001.

[6]. D. Paul and J. Baker, "The Design of Wall Street Journal-Based CSR Corpus," *Proceedings of ICSLP*, pp. 899-902, Banff, Alberta, Canada, October 1992.

[7]. J. Picone, *et al.*, "Speech Recognition System Training Workshop," ISIP, Mississippi State University, MS State, MS, USA, May 2002 (see http://www.isip.msstate.edu/conferences/srstw/).

[8]. J.P. Chin, V.A. Diehl, and K.L. Norman, "Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface," *Proceedings of SIGCHI'88*, pp. 213-218, New York, New York, USA, October 1988 (see http://lap.umd.edu/q7/quis.html*)*.