Tenth Quarterly Progress Report

January 1 through March 31, 2001

NIH Project N01-DC-8-2105

Speech Processors for Auditory Prostheses

Prepared by

Jeannie Cox, Robert Wolford, Reinhold Schatzer, Blake Wilson and Dewey Lawson

Center for Auditory Prosthesis Research
Research Triangle Institute
Research Triangle Park, NC 27709

CONTENTS

I. Introduction

The main objective of this project is to design, develop, and evaluate speech processors for implantable auditory prostheses. Ideally, such processors will represent the information content of speech in a way that can be perceived and utilized by implant patients. An additional objective is to record responses of the auditory nerve to a variety of electrical stimuli in studies with patients. Results from such recordings can provide important information on the physiological function of the nerve, on an electrode-by-electrode basis, and also can be used to evaluate the ability of speech processing strategies to produce desired spatial or temporal patterns of neural activity.

Work and activities in this quarter included:

- Studies with subject ME8 during the three-week period beginning on January 8. This subject was referred to us by our colleagues in Manchester, England, and is a recipient of a COMBI 40S implant on one side and a COMBI 40+ implant on the contralateral side (the 40S implant is designed for a short insertion and includes 8 rather than 12 electrodes). The studies included measures of sensitivities to interaural timing and amplitude differences and evaluation of various processing strategies either to represent cues for sound localization or to exploit the availability of bilateral implants in other ways (see Quarterly Progress Report 4 for this project, for a detailed discussion of processing options for bilateral implants).
- A visit by Martin O'Driscol of the Manchester Cochlear Implant Team, in conjunction with the visit by subject ME8. (Mr. O'Driscol visited us during the period from January 22 to 28 and participated in the studies with ME8.)
- Studies with subject ME9 during the two-week period beginning on March 5. This subject was referred to us by our colleagues in Würzburg, Germany, and is a recipient of COMBI 40+ implants on both sides. Studies with him were similar to those outlined above for subject ME8.
- A visit by consultant Marian Zerbi, to work with Reinhold Schatzer in the further development of the speech reception laboratory, especially monitor programs for the specification of speech processor designs in studies with recipients of bilateral cochlear implants (Jan 9-12).
- Participation by Lianne Cartee in a workshop sponsored by the Advanced Bionics Corporation, on a new interface system the company has developed for support of research studies with their CII implant device (January 26-29, in Los Angeles).
- Visits by three members of an Engineering Research Center on "Wireless Integrated Microsystems" at the University of Michigan (February 23). One of two demonstration projects for the Center is development of a fully-implanted cochlear prosthesis, and the group from the Center visited us to learn more about the design and implementation details for CIS processors. The visit was hosted by Reinhold Schatzer.
- Participation by Reinhold Schatzer in the *8th Symposium on Cochlear Implants in Children*, held in Los Angles, February 27 to March 3.
- Further development of tools for support of our studies, including the tools listed in the first paragraph below.
- Continued analysis of psychophysical, speech reception, and evoked potential data from current and prior studies.
- Continued preparation of manuscripts for publication.

In this report we describe the development and/or evaluation of new tools for our ongoing and future work. The tools include (1) application of speech material in the TIMIT speech database

for tests with implant patients, (2) software to process speech and other sounds using head-related transfer functions, and (3) a Microsoft Access database of speech processor designs and study results.

Evaluation of the TIMIT database was conducted by Bob Wolford. He also designed tests for our studies using selected items and lists of items from the database. Reinhold Schatzer developed the software for processing inputs using head-related transfer functions. The Access database was designed by Jeannie Cox, Bob Wolford and Marian Zerbi, with assistance from other members of our team. Jeannie Cox populated the database once its initial design was completed. She continues to refine the design and to maintain the database with new entries as results from ongoing studies become available.

Results from other studies and activities listed above will be presented in future reports.

II. Overview of New Tools

We have used a variety of both commercially-available and custom speech test materials to document the speech reception abilities of our implant subjects when using different types of speech processors. Each has undergone a thorough evaluation by us to assess its appropriateness and reliability for measuring performance in terms that will allow comparisons across subjects, implanted devices, and research groups. Some tests and materials include:

- VCV (medial consonant) tokens for use by speakers of English, German, Italian, and Spanish, including both 16- and 24-token tests for the English case
- CV tokens (16- and 23-token tests)
- Iowa Test of Vowel Recognition
- Cochlear Corporation recordings of the NU6 word lists
- CNC word lists
- MRT single and multi-talker word lists
- CID W-22 Word lists
- Cochlear Corporation recordings of the CUNY sentence lists
- Starkey/House Ear Institute HRTF-processed HINT and HSM sentences
- All tests of the Minimal Auditory Capabilities (MAC) Battery
- Cochlear Corporation recordings of the CID sentences
- Synthetic Sentence ID
- Dichotic Sentence ID
- HINT sentences
- German HSM sentences
- German OLSA sentences

A comprehensive review of test procedures we have used (and in many cases developed) over the years is presented in Quarterly Progress Report 11 from our prior project (Lawson *et al.*, 1998). That report also presents results from cross-calibration of various materials, which serves as a useful guide to the suitability, reliability, and relative sensitivities of different materials for different purposes, *e.g.*, the relatively rapid and reliable screening of processor designs with 16- and 24-consonant tests, or the use of especially difficult material for comparisons among designs in studies with "high performance" subjects.

In the current project we evaluated use of the TIMIT speech database as an additional source of especially difficult material. One attractive feature of the database is that it provides a quite large supply of sentence material, produced by many talkers with various U. S. regional accents and dialects. The sentences, if suitable, would complement the use of other difficult material, in that the other material is limited to monosyllabic words and CV syllables (see Lawson *et al.*, 1998).

As described in detail below, our evaluation demonstrated the suitability and high sensitivity of the TIMIT sentences for measures of speech reception performance by subjects enjoying the best results with current speech processor designs. The TIMIT sentences have considerable "headroom" for measures of further gains in performance that might be achieved with new designs. The size of the database also allows within-subject comparisons of many processing alternatives, without exhausting the available materials.

Another need of the current project was to develop tests and test materials for evaluation of speech reception performance by subjects with bilateral implants. For this, we decided to use head-related transfer functions (HRTFs) to impart sound localization cues in recordings of speech

signals and noise. The apparent direction of incidence of both speech (the signal) and noise can be manipulated with HRTF processing. Processed recordings then can be presented directly to the stereo inputs of the processors under test, reducing or eliminating the need for free-field tests in a special room, with sources of signal and noise physically located in the desired directions.

For some years, we have used a hardware and software system made available to us by the House Ear Institute and the Starkey company for presentation of the HINT sentences, processed with HRTFs to place the signal at the front and a source of interfering speech-spectrum noise at the front or to either side. Sigfrid Soli and Michael Nilsson of the House Ear Institute graciously processed the German language HSM sentences for us in a similar way. This system works well but its use is limited to the English HINT and German HSM sentences. We needed HRTF-processed consonant identification materials and additional sentence materials for both English- and German-speaking subjects.

In response to this need we developed MATLAB scripts for processing a variety of speech and noise records using HRTFs produced by the Media Laboratory at the Massachusetts Institute of Technology. The tools and database for HRTF processing are described in detail below. We note that many of the materials used routinely by us for testing subjects with unilateral implants now have been processed, so that they also can be used in directionally-sensitive tests with recipients of bilateral implants. The processed materials include speech from the front, optionally combined with noise from the front, from 90 degrees to the right, or from 90 degrees to the left. Materials processed to date have included the medial 16- and 24-consonant test tokens, NU6 monosyllabic words, and the CUNY sentences.

A major effort was initiated in the current project to develop an Access database of speech processor designs and study results, to bring this information together in one place for fast access and in a structure that allows retrieval of prior designs and results on the basis of shared attributes and parameter values. The database includes a tremendous amount of information, in a uniform and searchable format. The task of designing and building the database was far from trivial, in part because we have accumulated records of more that 150 subject visits over the years and in part because the specification of speech processor designs has evolved over the years (*e.g.*, to accommodate new processor structures as they were developed or to accommodate use of existing designs with new types of implant devices). All of those changes are recognized and tabulated in the Access database. The database has proven itself to be an extraordinarily powerful tool for identifying and building upon prior processor designs and for bringing together data from certain classes of studies (*e.g.*, to evaluate effects of rate manipulations in conjunction with various types of speech processors) or certain classes of patients (*e.g.*, patients with a particular device or patients with a low- or high-level of speech reception performance). The facilitated access to data, and the ability to conduct structured analyses of identified data, already have been most helpful in the preparation of manuscripts and in the design of ongoing and future studies.

A full description of the database and its design is presented below. We plan to develop similar databases for the psychophysical and evoked potential studies conducted in our laboratories over the years. As with the speech reception studies, the conditions and data for these latter studies are quite large and diverse. Bringing those conditions and data together in a searchable database will be valuable. We expect to begin development of the separate databases for the psychophysical and the evoked potential studies in the next quarter.

III. Evaluation of the TIMIT Speech Database for Use in Studies with Implant Subjects

Happily, over the years we have been measuring speech understanding among cochlear implant users, there have been continuing improvements in both the best results and the average results among those users. As such improvements have occurred, changes have been needed in the speech materials used to document performance in both quiet and noise conditions, in order to maintain sensitivity to differences in performance among processing strategies. Medial consonant identification tests have proven to be a rapid measure of speech identification, with easily characterized statistical uncertainties and the freedom of repeat testing without significant contamination of subsequent results. These should, of course, be calibrated by and used in conjunction with open set tests such as single-syllable word identification and identification of words in sentences. However, there are only a limited number of well-balanced single-syllable word and sentence lists available. Word lists such as the NU6 and CNC contain highly memorable juxtapositions of words and thus are vulnerable to contamination of results by retesting with the same material. Sentence lists such as the CUNY also contain memorable phrases that raise serious questions about reuse with the same subject. In addition, at least four such lists must be presented in each tested condition in order to achieve a good enough balance of material to support the reliable detection of small performance differences. Further confounding the use of these materials is the fact that they are widely employed in clinical programming, making it almost impossible to know which sentence lists a research subject may have been exposed to previously.

To limit these possible contamination issues we have begun to incorporate use of the TIMIT sentences into our testing regime.

The TIMIT Sentences

The TIMIT corpus of speech read from scripts contains speech from 630 talkers (male and female) and from eight major dialects of American English, with each talker uttering ten phonetically-rich sentences. The TIMIT corpus was designed to provide speech data for the acquisition of acoustic/phonetic knowledge and for the development of automatic speech recognition systems. This work has resulted from the joint efforts of several sites under sponsorship from the Defense Advanced Research Projects Agency -- Information Science and Technology Office (DARPA-ISTO), including the Massachusetts Institute of Technology (MIT), Stanford Research Institute (SRI), and Texas Instruments (TI). The speech was recorded at TI, transcribed at MIT, and has been maintained, verified, and prepared for CD-ROM production by the National Institute of Standards and Technology (NIST).

Use of TIMIT Sentences for documenting performance levels of cochlear implant subjects

A primary role of speech testing materials in our research is the measurement of levels of performance that will allow comparisons across subjects, implanted devices, and research groups. Accordingly, we set about the process of selecting and validating the use of the TIMIT sentences for this purpose. As noted above, we previously completed a similar cross-comparison in our laboratory for the tests now being used for comparison with the TIMIT sentences (Lawson, *et al*., 1998). An observation at that time was a number of implant recipients who demonstrated ceiling effects for tests routinely used to document implant performance. Fortunately, this trend has continued and the number of individuals with significant benefit from cochlear implantation has

continued to grow with a proportionate limitation of test materials due to ceiling effects. The present cross-comparison will show the difficulty level of the TIMIT sentences in quiet to be higher than the other tests, even when the other tests are presented in noise, at the tested speech-to-noise ratios. Our validation of the selected TIMIT sentences also demonstrates that their use provides a reliable measure for describing speech understanding of cochlear implant recipients who are currently near the top end of the performance spectrum.

The TIMIT sentences were sorted and grouped, after elimination of sentences that were repeated within the database by several different speakers. This subset then was transformed to files in the *.wav format, for easy playback in our laboratory environment. The selected sentences were grouped in lists of similar length and difficulty. The result is 50 lists containing 20 sentences each, for a total of 1000 sentences. Each list contains sentences by both male and female speakers, spoken with different vocal effort and from different dialectical regions of the United States. Score sheets have been created, providing word counts for each list.

The sentences were first screened for audibility then combined with other speech testing measures during evaluations of speech processor designs in tests with cochlear implant subjects. Design variables that were being investigated at the time of this analysis included the effect of changes in rate of stimulation, compression function, or manipulations of both rate and the cutoff frequency setting of the low pass filter in the envelope detectors of CIS processors. The speech measures used in these comparisons included the identification of medial consonant (VCV) tokens (both male and female speakers), identification of medial vowels (hVd), identification of monosyllabic words from CNC lists, identification of words in CUNY sentences and, of course, identification of words in sentences from the newly created TIMIT lists. Some of the materials also were tested in different levels of noise. Not every measure was evaluated in each condition or across every subject.

Figure III.1 shows results for two subjects who were tested with CIS speech processors using various rates of stimulation. The measures included identification or recognition of the Male VCV, Female VCV, CUNY sentences, CNC words and the TIMIT sentences, all in quiet. In addition, subject SR-2 completed the VCV, CNC and CUNY tests at a +15 dB signal-to-noise ratio.

Within the data for each of these subjects, there is obvious correlation among the results of the various tests, with the exception of the CUNY sentence results for subject SR-3, which are limited by ceiling effects.

Figure III.2 shows comparisons for the same two subjects across six CIS processors using different exponents for the compression function. The results of the TIMIT tests in quiet and in noise, while significantly lower than the scores for the other tests, indicate the same relative performance rankings among the tested processors.

In Fig. III.3, TIMIT sentence scores are compared to those from male medial consonant identification tests completed in quiet, and at +10 and +5 dB signal-to-noise ratios, for processors with various combinations of stimulation rate and envelope smoothing filter settings. These nine different processors all show lower scores with the TIMIT test in quiet than for the medial consonant tests at either of the signal-to-noise ratios, but again indicate the same relative performance relationships across the processors.

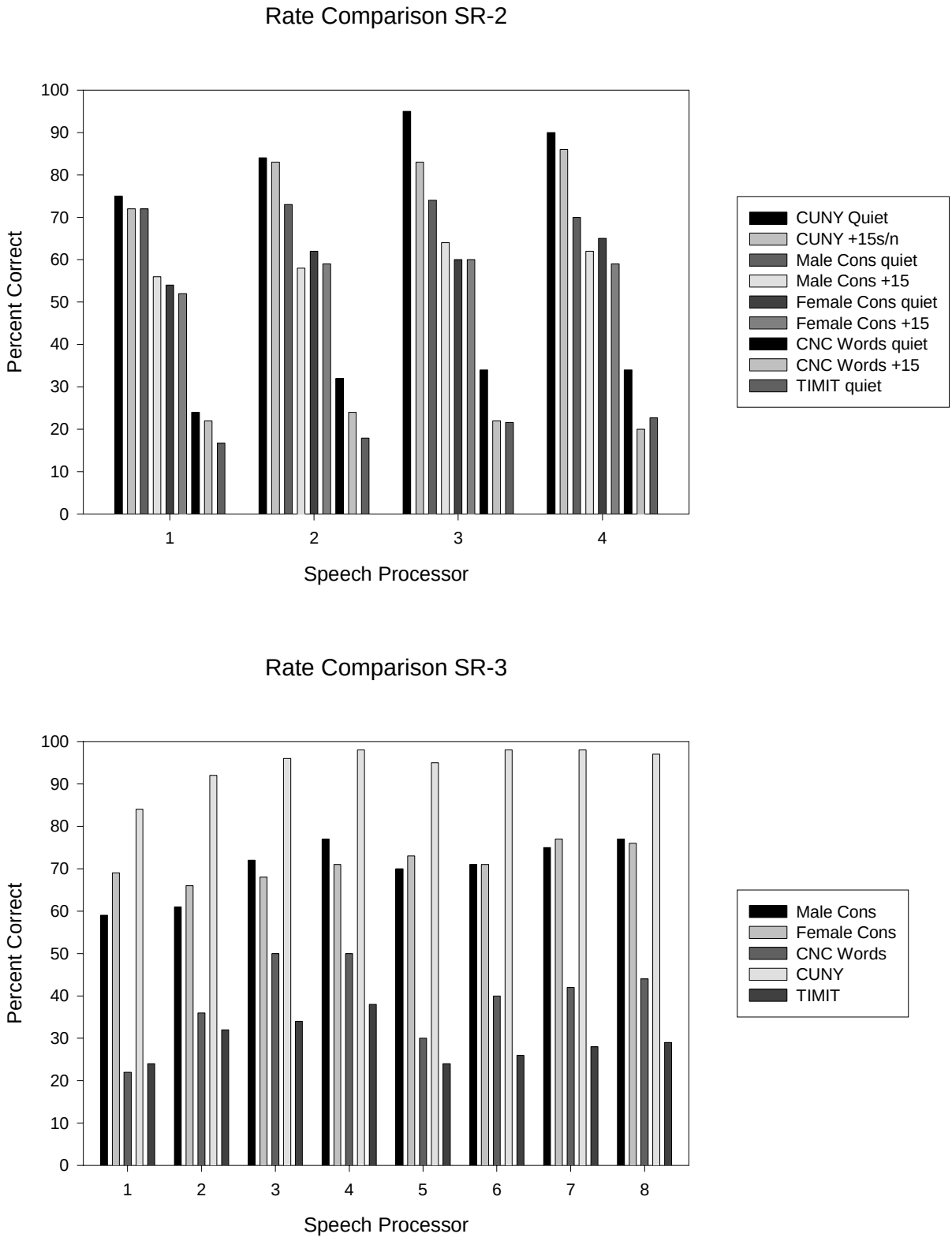

Fig. III.1. TIMIT sentence word identification scores are compared with scores from identification of male and female medial consonant tokens, identification of CNC monosyllabic words and identification of words in CUNY sentences. The data are for two subjects tested with CIS speech processors using various rates of stimulation. Subject SR-2 was tested at a signal-to-noise ratio of +15 dB, as well as in quiet.

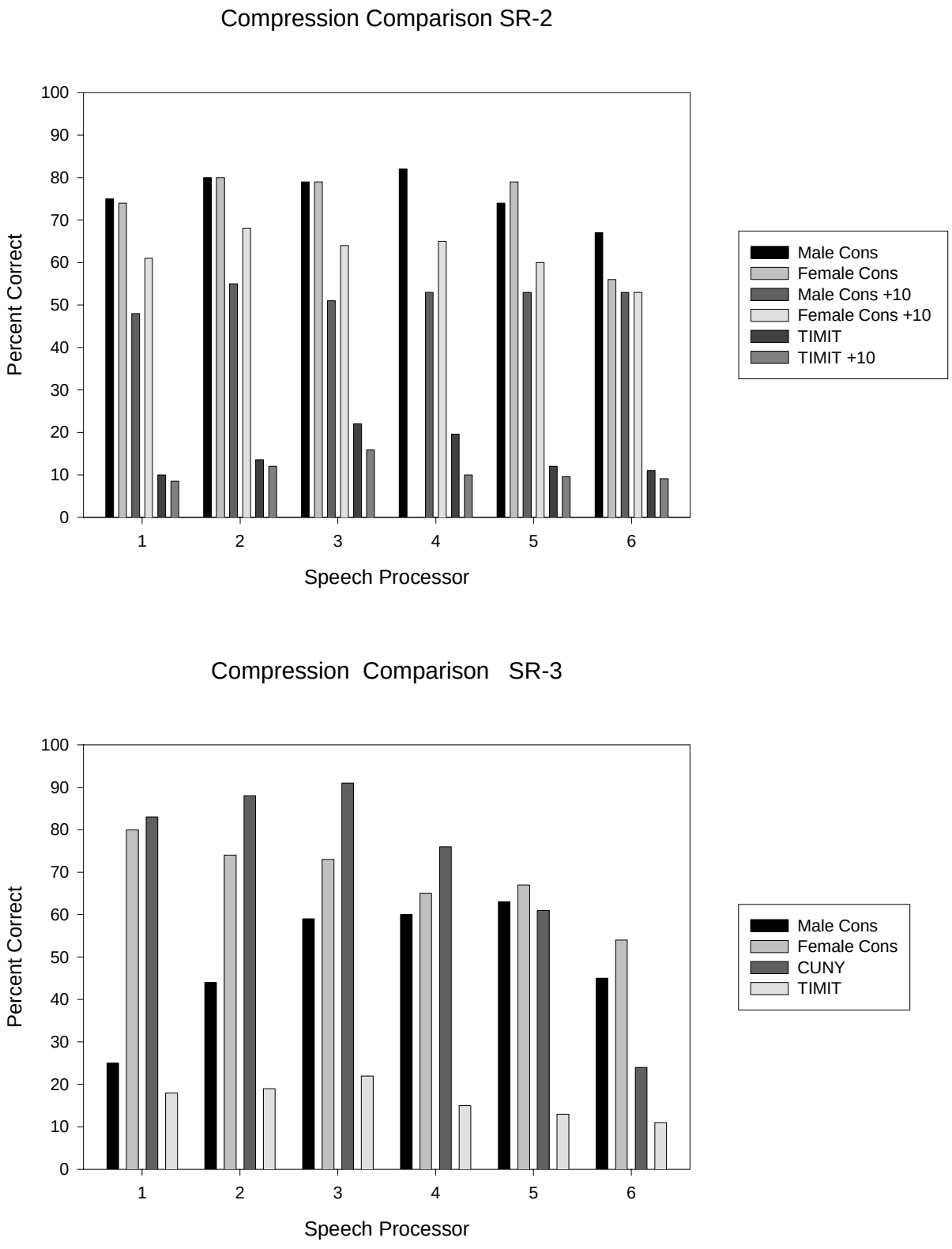

Fig. III.2. TIMIT sentence word identification scores are compared with scores from identification of male and female medial consonant tokens, and identification of words in CUNY sentences. The data are for two subjects tested with CIS speech processors based on a variety of different compression functions. Subject SR-2 was tested at a signal-to-noise ratio of +10 dB, as well as in quiet.

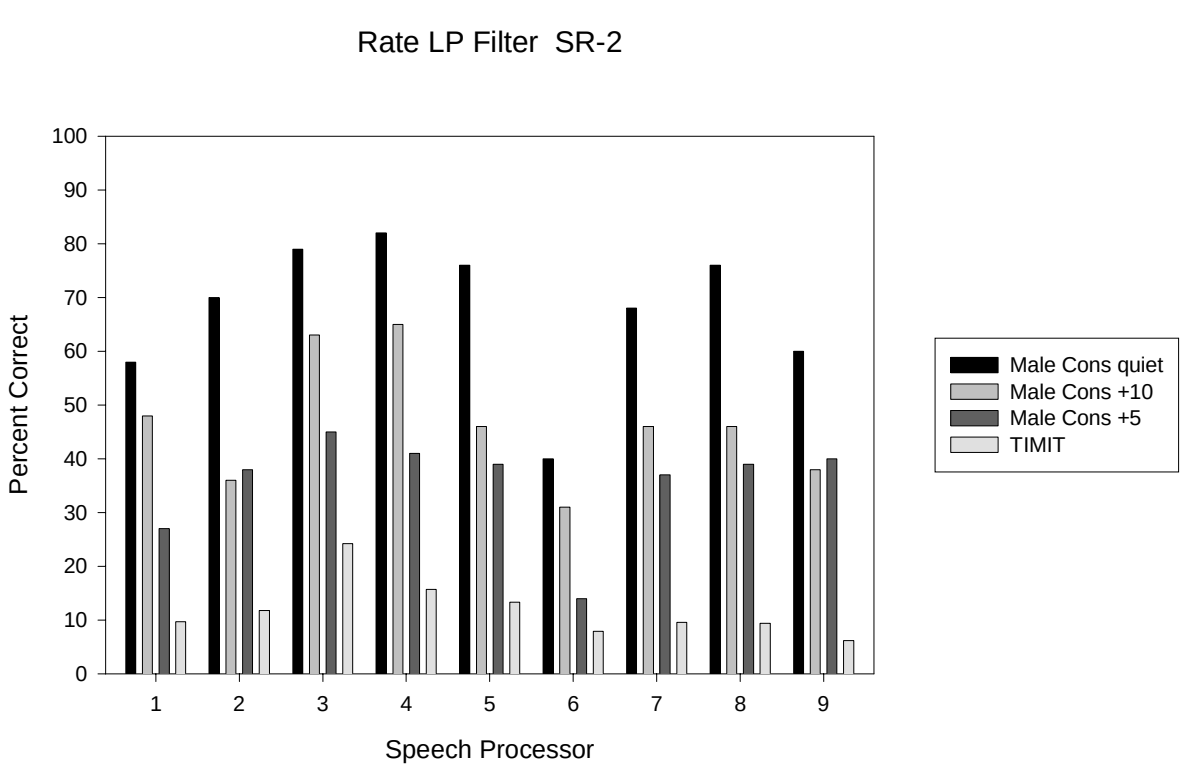

Fig. III.3. TIMIT sentence word identification scores in quiet are compared with scores from identification of male medial consonant tokens in quiet and at the signal-to-noise ratios of +10 and +5 dB. The data are for subject SR-2, tested with CIS speech processors having various combinations of rate and smoothing filter settings.

The above results demonstrate that tests using sentences selected from the TIMIT speech database, while being more difficult than the other tests, provide reliable and sensitive measures of speech recognition for cochlear implant subjects. *Post hoc* evaluation of the results indicates that the minimum number of lists that need to be administered for each condition is three. This number of lists provides an average word count of approximately 408 words, which is the same count previously found necessary for cross-test comparison sensitivity using CUNY sentences.

At present, the level of difficulty of the TIMIT test restricts its useful application to individuals whose performance levels are in the top half of overall implant results. With the exception of the CNC word test scores in Fig. III.1, the results of the TIMIT test in quiet yields lower scores than each of the other tests in this analysis, including presentations of test material in competition with noise. As implant performance continues to improve, however, the TIMIT test should be useful for an increasing fraction of our subjects, and we plan to continue this type of investigation and development of additional measures to validate implant performance.

IV. Processing of Speech and Other Sounds Using Head-Related Transfer Functions

New tools have been developed to process speech materials for use in studies with recipients of bilateral cochlear implants. The tools are implemented in MATLAB and include two custom scripts for use in the MATLAB environment.

The first of these scripts convolves an input file (in *.wav format) with a selected head-related transfer function (HRTF) to simulate an incidence of source at a particular azimuth and elevation. A stereo output is produced, for binaural presentation to the subject. The second script mixes two binaurally-processed stereo wave files, typically a speech signal with concurrent noise, at a given signal-to-noise ratio (SNR).

In the remainder of this section we describe the procedures that are implemented with the two scripts. Listings of the scripts are presented in Appendix 1 to this report. The Appendix also includes listings of two helper functions that are utilized by the custom scripts. The helper functions are public-domain scripts provided by the MIT Media Laboratory.

The custom scripts are based on data from a set of HRTF measurements made with a KEMAR manikin. The data were collected by Bill Gardner and Keith Martin at the MIT Media Laboratory. These data are Copyright 1994 and have been made available to the research community on the Internet via anonymous FTP and the World Wide Web. The HRTF material as well as a detailed description of the measurement technique can be found at http://sound.media.mit.edu/KEMAR.html.

The MATLAB tools were used to generate speech test materials aimed at demonstrating possible advantages of bilateral cochlear implants versus unilateral implants. Medial consonant test tokens, NU6 monosyllables and CUNY sentences were processed with the HRTF filters and then mixed with CCITT speech-spectrum noise at SNRs from –10 to +15 dB, in 5dB steps. In all processed materials, signal incidence is frontal (azimuth = 0 degrees and elevation = 0 degrees), while noise was added with incidence in the horizontal plane from left (azimuth = 270 and elevation = 0 degrees), front (azimuth = 0 and elevation = 0 degrees), and right (azimuth = 90 and elevation = 0 degrees). These test materials are particularly helpful in quantifying and separating head shadow from binaural summation and squelch effects (if present) in bilaterally implanted subjects.

MIT HRTF data

The MIT ftp archive contains:

- Full set of HRTF measurements, including speaker and headphone responses
- Compact HRTF measurement set
- Speaker and headphone responses
- Diffuse-field equalized HRTFs (44.1 kHz)
- Diffuse-field equalized HRTFs, re-sampled to 32kHz sampling rate
- Useful MATLAB scripts for processing data (including the helper files mentioned above)
- Documentation files

Data in the "full" data set are un-equalized, and thus contain the frequency response of the measurement system, including the speaker, microphone, and electronics.

The compact data are equalized using a minimum-phase inverse filter of the loudspeaker's response (a Realistic Optimus Pro 7 speaker was used). The speaker's response was measured with a Neumann KMi 84 microphone (cardioid). Thus, the compact data may contain phase aberrations of the measurement system that are not necessarily part of the HRTFs.

Both the full data and compact data include the ear canal resonance. When these HRTFs are used in an auditory display, the listener will hear the KEMAR ear canal resonance (in addition to his/her own ear canal resonance for normal-hearing listeners). To factor out both the measurement system response and ear canal response, the measurements were normalized with respect to an average across all directions (diffuse-field equalization). The diffuse-field equalized data set was created by forming the power spectrum average across all incident directions, inverting this using a minimum-phase inverse filter, and applying the resulting filter to the data set. These data are sampled at 44.1 kHz and 32 kHz.

Description of MATLAB scripts

HRTF processing and signal (noise) mixing is implemented in the two MATLAB scripts *DoHRTF.m* and *Mixer.m*, respectively. ***DoHRTF.m*** takes a monophonic input signal *x(n)* from a wave file and convolves it with the appropriate pair of HRTFs to make the resulting signal (presented binaurally) appear to come from a sound source at a given azimuth $0 <= \phi < 360$ and elevation $-40 <= \theta <= 90$ (see Fig. IV.1 below). *DoHRTF.m* uses the diffuse-field equalized data to compensate for speaker and KEMAR ear canal responses.

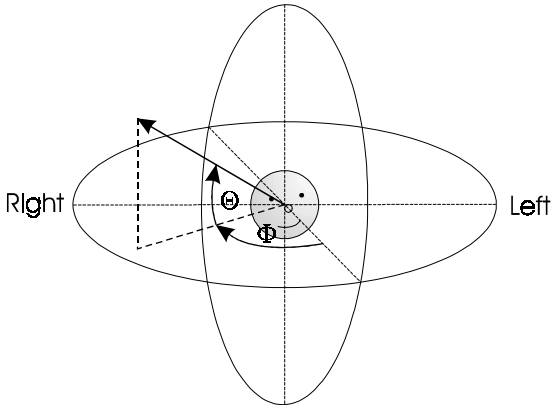

Fig. IV.1. Frame of reference for definition of azimuth ϕ and elevation θ.

Thus, with the appropriate pair of HRTF impulse responses $h_L^{\Phi,\Theta}(n)$ and $h_R^{\Phi,\Theta}(n)$ for left and right ear and sound incidence from azimuth ϕ and elevation θ, the left and right output signals $y_L^{\Phi,\Theta}(n)$ and $y_R^{\Phi,\Theta}(n)$ are given as

$$\begin{aligned} y_L^{\Phi,\Theta}(n) &= x(n) * h_L^{\Phi,\Theta}(n) \\ y_R^{\Phi,\Theta}(n) &= x(n) * h_R^{\Phi,\Theta}(n) \end{aligned} \quad \text{for } 0 \le \Phi < 360, \qquad \textbf{Equation 1}$$

where * denotes the digital convolution operation

$$y(n) = x(n) * h(n) := \sum_{i=0}^{N-1} x(n-i)\, h(i). \qquad \textbf{Equation 2}$$

Due to the implicit symmetry of the diffuse-field HRTF data (those data are available for azimuth angles 0 <= ϕ < 180 only), we have to exchange $h_L^{\Phi,\Theta}(n)$ and $h_R^{\Phi,\Theta}(n)$ in **Equation 1** to get the left and right output signals $y_L^{\Phi,\Theta}(n)$ and $y_R^{\Phi,\Theta}(n)$ for azimuth angles 180 < ϕ < 360:

$$\begin{aligned} y_L^{\Phi,\Theta}(n) &= x(n) * h_L^{\Phi,\Theta}(n) \\ y_R^{\Phi,\Theta}(n) &= x(n) * h_R^{\Phi,\Theta}(n) \end{aligned} \quad \text{for } 0 \le \Phi \le 180, \qquad \textbf{Equation 3}$$

$$\begin{aligned} y_L^{\Phi,\Theta}(n) &= x(n) * h_R^{\Phi,\Theta}(n) \\ y_R^{\Phi,\Theta}(n) &= x(n) * h_L^{\Phi,\Theta}(n) \end{aligned} \quad \text{for } 180 < \Phi < 360. \qquad \textbf{Equation 4}$$

The algorithm implemented in *DoHRTF.m* for binaural processing is shown in Fig. IV.2.

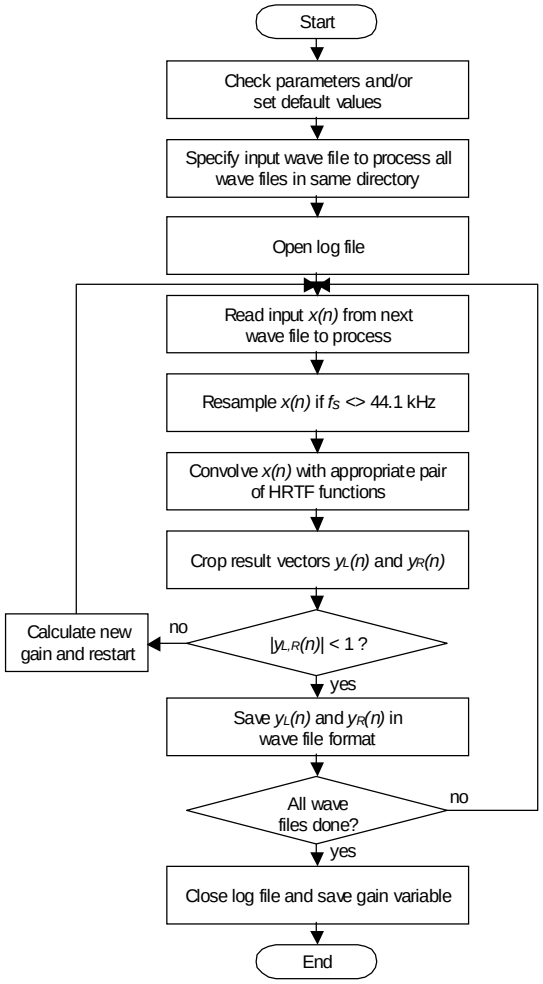

Fig. IV.2. Flowchart of MATLAB script *DoHRTF.m*

DoHRTF is invoked from the MATLAB command line by optionally providing parameter values in the function call. A syntax help text will be displayed if "`help dohrtf`" is issued. If no parameters are provided in the function call, the script sets most parameters to default values (*e.g.*, the default for azimuth and elevation angles is 0 degrees for both). *DoHRTF* then opens a file requester and asks the user to specify an input wave file. All wave files contained in the same directory as the file specified will then be processed and the result saved in a subdirectory named HRTF. If not specified in the function call, the first time a stereo input file is encountered the user will be asked to select which side to process (the input signal *x(n)* must be monophonic). The same side will be processed for all stereo files in the current batch. The output wave file names are composed by attaching "`_EeAAAa.wav`" to the input file names, where `E` is substituted for the elevation and `AAA` for the azimuth angle in degrees. For example, the input wave file `aba.wav` processed for sound incidence from elevation 0 and azimuth 90 degrees, is saved as `aba_0e090a.wav` in the subdirectory HRTF.

Each processing step is displayed on the screen and commented into a log file `logfile.txt`, which is stored in the HRTF subdirectory together with the processed output files.

If the input wave file is not sampled at 44.1 kHz, *DoHRTF* tries to resample it. This works if the input file's sampling rate is an integer fraction of 44.1 kHz. If it is not, the input file is skipped.

The signal vector *x(n)* is then convolved with the appropriate pair of diffuse-field equalized HRTF functions, according to **Equation 3** or **Equation 4**, for a given azimuth angle ϕ. Since the used HRTF vectors contain 128 samples, the convolution result is length(*x(n)*)+127 samples in length. In order for the output signals $y_L^{\Phi,\Theta}(n)$ and $y_R^{\Phi,\Theta}(n)$ to have the same length as the input signal *x(n)*, the leading 64 and trailing 63 samples in the convolution results are dropped. Next, the output signals are checked for clipping (samples are 16 bit signed fractionals in the range [-1, 1]). If samples are found to be out of range, all input wave files are re-processed (for all azimuth/elevation angles specified) with a slightly decreased overall gain factor to avoid clipping. Re-processing of all wave files with the same digital gain factor ensures that loudness balanced input wave files will remain balanced after HRTF processing.

When all input wave files are processed, the log file is closed, and the final overall gain factor is saved from the MATLAB workspace into the file `gain.mat` in the HRTF subdirectory.

Mixer.m takes two HRTF processed stereo input wave files at a time from two separate directories and combines them at a specified amplitude ratio. Since typically speech signals (at frontal incidence) are mixed with noise (coming from various directions), the mixing ratio is specified in dB SNR. A syntax help for *Mixer.m* can be obtained by issuing "`help mixer`" at the MATLAB command line.

Figure IV.3 shows a flowchart of *Mixer.m*.

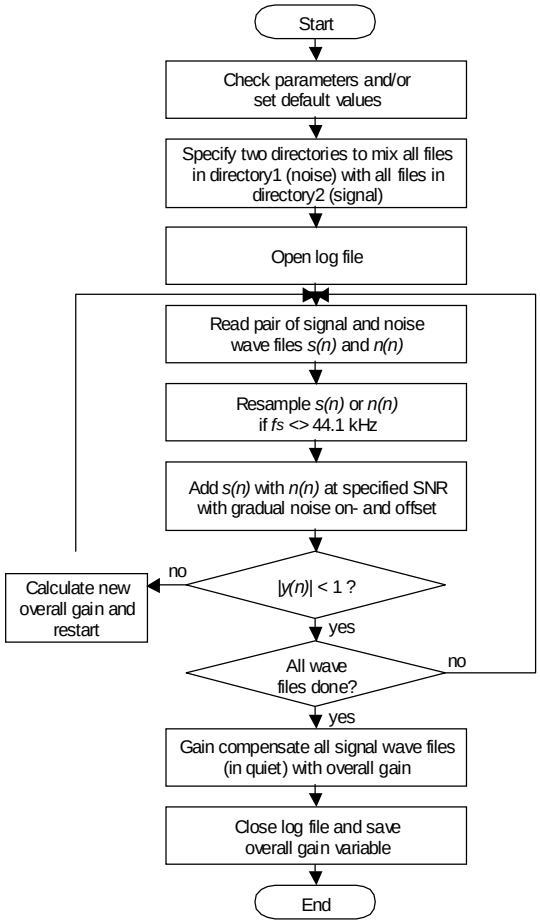

Fig. IV.3. Flowchart of MATLAB script *Mixer.m*

At first, *Mixer* checks arguments provided in the function call, and sets default values for omitted parameters. If not specified explicitly, the program uses a default signal to noise calibration factor, which has been experimentally determined by calibrating a CCITT noise wave file played through our speechlab's PC soundcard with a 1000Hz test tone. This same CCITT noise wave file was used to generate all binaural test materials so far, and had been widely used in previous monaural work in our laboratory.

Like *DoHRTF*, *Mixer* works on a directory basis, *i.e.*, all wave files in the "noise" directory are mixed with all wave files in the "signal" directory. The output wave files are stored in a unique subdirectory for each SNR at which the input waveforms are mixed together .

The stereo signal and noise waveform vectors retrieved from a pair of wave files are added at relative amplitudes defined by the given SNR and by taking into account the calibration factor mentioned above. Signal onset is delayed by 1.8 seconds with respect to noise, with the noise amplitude being gradually increased from 0 to 100% over the first 1.5 seconds. The noise waveform extends beyond the end of the signal in each case in a way that is symmetrical to the noise onset.

Next, the mixed signal is checked for clipping. If samples are found to be out of range, mixing of all wave files is restarted with an adjusted value for the overall mixed signal gain. This iterative procedure terminates when all signal/noise wave file combinations are processed without clipping. The final overall gain factor is then applied to all signal wave files to loudness balance the speech signals in quiet with the ones with noise added. As in *DoHRTF*, the overall gain is documented in the log file and saved from the MATLAB workspace into a file `gain.mat`.

V. Access Database of Speech Processor Designs and Study Results

Our group has accumulated records of more than 150 subject visits since 1988. Those records take a variety of physical forms, from hand notations on paper to information in a variety of formats on computer disk. Examples of the range of data types include hearing histories, daily sequences of events, precise documentation of each of thousands of processor designs tested, sets of threshold and MCL stimulation levels for various combinations of electrodes and stimulation waveforms, records of every individual response in speech tests (identification of consonants and vowels, identification of monosyllabic words and words in sentences, etc.), detailed results for a variety of psychophysical tests (pitch ranking, pitch scaling, loudness scaling, lateralization, localization, masking, gap detection, etc.), and voluminous recordings of physiological signals, including scalp potentials and intracochlear evoked potentials.

Although the results and experiences of each visit are well documented, the variety of media and data storage methods result in records that are time-consuming and cumbersome to search. To improve the efficiency of certain types of searches, considerable time was spent entering data into an Excel spreadsheet, both manually and from reading processor-specification files originally recorded on floppy disks. The data acquired during subject visits were manually entered into the spreadsheet for storage and later comparison.

This approach was limited. The data were difficult to sort and could be searched only on the basis of a highly restricted set of criteria. Because data had to be transcribed from one source to another, additional possibilities for error were introduced. The limitations of the spreadsheet format prevented much wider use of a wealth of information. To increase the utility of the data we have collected over many years of studies, we decided to incorporate them into a database structure, that would allow flexible and rapid searches for particular types of information and also would minimize the possibilities for error inherent in the multiple formats previously used for recording data.

We began by addressing each of the problems and limitations of our previous data storage methods and designing an alternative that could benefit from the most up-to-date computer technology.

Prior formats for data storage and retrieval

Our early data storage was primitive by today's standards. At the beginning of our work (starting in 1983), we recorded speech test conditions and results in laboratory notebooks. Specification files for speech processor designs were recorded on computer tapes for the first several years and then were recorded on floppy disks. The specification files included information on stimulation rate, band pass filter type, electrodes selected for stimulation, stimulation order, overall gain, documentation of the underlying psychophysical results in each case, and other parameters. Related paper records for subject testing included identification of processor type (*e.g.*, interleaved pulses or compressed analog) and the laboratory configuration for each test (processing hardware, monitor software, audio channel inputs and other parameter settings).

Results from tests of consonant identification were archived separately in text files. These files were composed of one ASCII text line per test that constituted the archival record of the conditions and individual responses to each consonant token presented.

It became evident that matching subject results to a particular speech processor design needed to be much easier. As the number and length of subject visits increased, information about speech processor design and test results was entered manually into spreadsheets for each visit by a subject (initially Lotus 1-2-3 spreadsheets and later Excel spreadsheets). Although this allowed more data to be viewed together, across-subject comparisons of results with similar speech processor designs remained unwieldy. Searching was limited and incomplete due to the inaccessibility of some information, inappropriateness of spreadsheet technology for linking to important classes of data, evolution of data categories (with resulting inconsistencies in some data), and the amount of time required to enter and search records. Copying data from one format to another also required multiple checks and rechecks to ensure accuracy.

In recent years we have used two formats for the specification of speech-processor designs. The first is for processors implemented with an older laboratory system using two Motorola digital signal processors (DSPs). This format is called the **DSP** specification file format. The second format is for processors implemented with a newer and more flexible laboratory system based on the Motorola **A**pplication **D**evelopment **S**ystem (ADS), which uses a single but much faster DSP. This format is called the ADS specification file format.

The database

Microsoft (MS) Access was selected because it is user-friendly and powerful. To date, we have incorporated information contained in the prior Excel spreadsheets along with some of the associated speech processor specification files. This first merge was a significant accomplishment in that it required input from multiple formats and instantly provided a check of the manually-entered data against the actual specification files used to run the speech processor.

Implementation and design details

Work has proceeded in stages to preserve functionality throughout the process. Each stage has been and will be followed by thorough cross-checks to ensure accuracy and consistency. The first stage of design and data input has centered around the sets of data already available.

In order to make the current body of information available for searching, tables were designed for each form of input. The tables include ones for

- Subject information
- Spreadsheet information (for checking against other information, since the spreadsheet has been updated more frequently than other sources)
- Test results
- Speech processor specification files, for both monaural (DSP and ADS formats) and binaural processors (ADS format)
- Archived test records.

Detailed field information for each of these tables is shown in Table V.1. Because the data were saved initially using the various formats described above, some reformatting had to occur prior to populating the tables. The spreadsheet was imported into Access directly and the subjects table was small enough to be created by hand. However, the unusual formatting combined with the sheer bulk of information in the specification and other files required development of custom programs for reformatting.

Figure V.1 shows such a program. This program was written in C++. It filters a directory of subject information (in this case SR2) for specification files (here files without an extension) and reformatted those into tab-delimited files that Access can import (here named summary.txt). It also checks for errors and saves the names of any questionable files for further investigation. The before and after layouts of a file processed in this manner are shown in Figs. V.2 and V.3, respectively. Thousands of specification files have been imported using this procedure. The present program reformats monaural DSP specification files. Additional programs are under development for reformatting binaural DSP specification files and all ADS specification files, as well as archived test records.

An overview of the relationships among the initial tables is shown in Fig. V.4. For simplicity, the specification file tables have been merged into one line in the diagram, inasmuch as they each have the same relationships. Each spec files table is related to the spreadsheet, speech test results, and archived test records tables by a one-to-many relationship, because one processor implementation can be associated with more than one test. The table for subjects is related to the specification file tables and spreadsheet through a one-to-many relationship, because one subject can have multiple spec files. The table for speech test results is related to the archived test results and spreadsheet tables by a one-to-one relationship, because there is a separate entry in each table per test, although the tables do not all include the same tests (for instance, archived test results contains consonant test information only). All information contained in the tables for Archived test results and for spreadsheet results will eventually be moved into the table for speech test results.

Ongoing and future development

The primary goal of the initial effort has been to merge the current information into one consistent database. There is still some redundancy, in that in many cases the same set of information is in more than one place. For instance, for one patient's processor, the specification filename can be found in the specification files and the spreadsheet, as well as other sources of test data. The redundancy allows the tables to be matched, but also requires checking for spelling and slight naming differences. Duplicate data such as test results in the spreadsheet and speech test results tables will be addressed in the second phase of design.

Once the database includes all current data and has been thoroughly checked, the tables can be collected and normalized to eliminate redundancies. Figure V.5 shows the major tables of the system design presently under development This design includes several new tables, including ones to (1) validate associations of software monitors and hardware interfaces, (2) store bandpass and lowpass filter coefficients, and (3) store threshold and most-comfortable-loudness data for each subject and for various pulse rates, durations, electrodes and other parameters of stimulation.

In this new structure, the tables for binaural and monaural speech processors are kept separate for easy classification as well as to avoid many blank fields that would be produced by merging the two. Binaural files include information for the processors on each side, whereas these extra fields are not applicable to monaural processors.

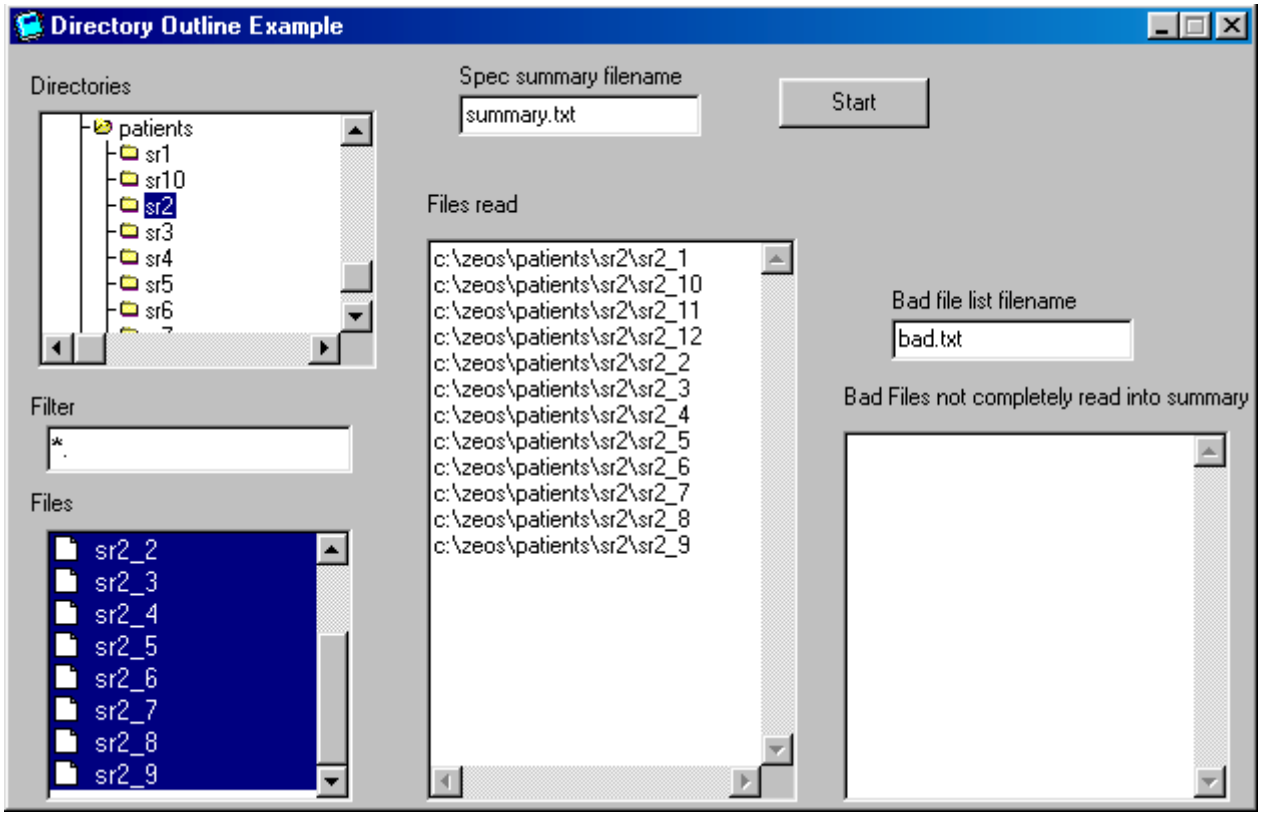

Fig. V.1. User interface of the program which reformats DSP specification files for importing into the Speech Processors Database.

Table V.1. Table names and descriptions for the initial version of the Speech Processors Database. Unique primary keys for each table are in bold italics and descriptions for selected fields are presented in parentheses.

Spreadsheet	Subject	Speech test results
implant type	patient ID	specfilename
patient ID	subject's name	test type
processor (sometimes has more information than standard spec files, such as n for noise)	patient initials	options
specfilename (combination of patient ID + processor)	implant type	sequence number
master gain		talker gender
seq num (unique sequence number for each processor and test combination)		date
channels (number of channels)		master gain
bp order (bandpass filter order)		noise
min f (minimum bandpass frequency)		presentations
max f (maximum bandpass frequency)		score
EQ		uncertainty or phoneme score

rct H_F (rectification: full wave or half wave)
smoothing filter (smoothing filter frequency)
smoothing order (smoothing filter order)
map (mapping law)
int master gain (internal master gain)
pulse rate
pulse width
leading phase
order of stim (order of stimulation)
electrodes
dyn range min (dynamic range - minimum)
dyn range max (dynamic range - maximum)
psychphys date (date psychophysics completed)
thres db (threshold in decibels)
MCL (most comfortable loudness)
m cons rep (male talker 16 consonant repetitions)
m16c (percent correct)
f cons rep (female talker 16 consonant repetitions)
f16c (percent correct)
m cons rep2 (male talker 24 consonant repetitions)
m24c (percent correct)
f cons rep2 (female talker 24 consonant repetitions)
f24c (percent correct)
NU6 word (NU6 word score)
nu6 phoneme (NU6 phoneme score)
question
notes
male vowel (male talker vowel repetitions)
mv (percent correct)
fem v (female talker vowel repetitions)
fv (percent correct)
CUNY (sentence word score)
CID (sentence word score)
Spondee (word score)
SPIN (sentence word score)
SPIN/lastwd (sentence last word score)
HINT (sentence word score)
CNC word (word score)
CNC phoneme (phoneme score)
TIMIT (sentence percent correct word score)

Monaural DSP spec files	Monaural ADS spec files
specfilename	specfilename
path (where file is saved)	date
file date (date of last changes)	system
nofm	load files
1dsp (primary processing and filter information)	input
2dsp (file with additional processing info)	channels

map	rate
mcl	pw (pulse width)
thresholds	psycho
chdef	order
globalgain	mode
pulsewidth	conditioner
orderofstim	map
mapexponent	gain
constantstim (boolean for constant stimulation)	delay
xymemory (additional information written in hex)	sync
numofchannels	extra (additional information)
vcis (boolean for virtual channels)	
rate	
comments	
filter	
calibration table	

Binaural DSP spec files	Binaural ADS spec files	Archived test records
specfilename	specfilename	patient initials
system	date	date
path	system	start time
date	cal_left (calibration file used for left ear)	sequence number
nofm	cal_right (calibration file used for right ear)	specfilename
cal_left (left ear calibration)	1processor	test type
cal_right(right ear calibration)	filter	talker gender (with or without vowels)
1proc	2processor	visual clues switch
filter	channels	audible signal switch
dsp2	rate	randomization code
maplawgain	pw	response (a string consisting of every response)
numofchannels	psycho	score (score correct)
mcl	order	
thresholds	conditioner	
chdef	cond rate	
globalgain	cond level	
modeofstim	map	
rate		
pulsewidth		
channel comp		
orderofstim		
mapexponent		
conststim		
comments		
xymemwrites		

```
*sr2_440
* Specification file
*
* 3 channel processor with 6th order bandpass and 400Hz 2nd order smoother
* full-wave rectification
* 16usec at 10400Hz rate MONOPOLAR positive PHASE 1ST
*
b:24cs3f4.lod
2
b:nd2deq.lod
*factor for maplaw's outputs - double
1.0
*number of channels
3
*mcl from 1 to 7 - double
822.26
978.85
1215.56
*thresholds - double
58.5
50.4
60.3
*define channel pairs 1 thru n
1,8
3,8
5,8
*global gain  x:$9 - decimal integer
4095
*pulse width x:$c and x:$d - integer
16
*order for channels
2
1
3
* lets define power function exponential for mapping
* defined in map.pwr[1] through map.pwr1[6] order
* setting all channels to map 3
-0.0001
*gain for 6th channel set to 1
x a fff
X C 7
```

Fig. V.2. Specification file sr2_440 prior to being formatted for the Speech Processors Database.

sr2_440		c:\zeos\patients\sr2\		10/11/1996			24cs3f4.lod		nd2deq.lod
	1.0	822.26, 978.85, 1215.56		58.5, 50.4, 60.3		1,8; 3,8; 5,8	4095	16	2, 1, 3
	-0.0001	x a fff; X C 7		3					

Fig. V.3. Specification file sr2_440 formatted into a tab delimited format, to be read into fields prepared in Access (see Table V.1).

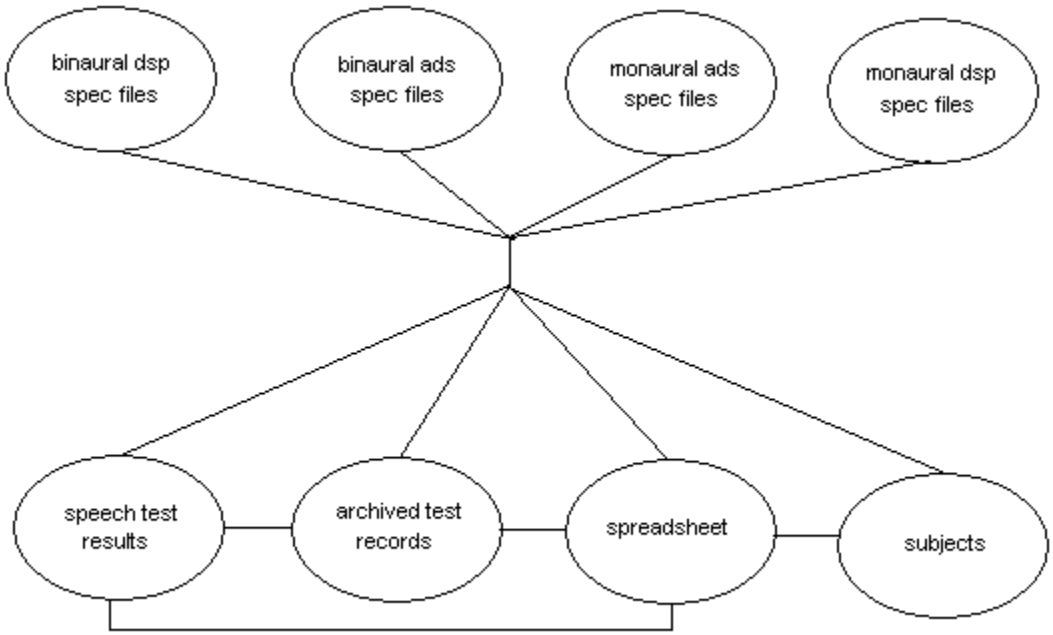

Fig. V.4. Overview of the present Speech Processors Database.

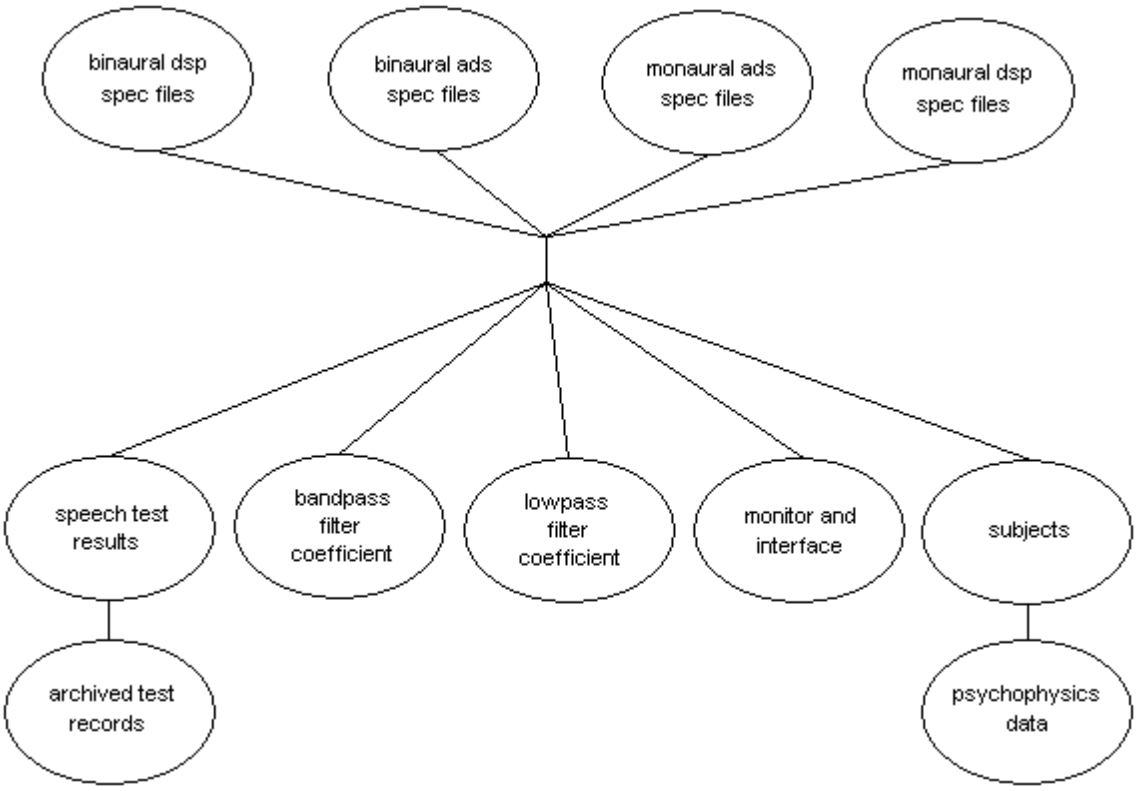

Fig. V.5. Major tables of the updated Speech Processors Database.

VI. Plans for the Next Quarter

Our plans for the next quarter include the following:

- Further development of the Speech Processors Database, as described in section V of this report.
- Initial development of databases for psychophysical and evoked potential studies.
- Studies with subject MI6, a low-performance non-user of the Clarion device, with some residual hearing on the side contralateral to the implant (April 16-20).
- Presentations of invited lectures by Stefan Brill and by Blake Wilson at the Wullstein Symposium, to be held in Würzburg, Germany, April 26-30, 2001 (this symposium includes the 2nd Conference on Bilateral Cochlear Implantation and Signal Processing, the 6th International Cochlear Implant Workshop, and the 2nd Auditory Brainstem Implant (ABI) Workshop; Wilson will be a Guest of Honor at the Symposium).
- Studies with subject ME6, in a return visit by this subject (present studies are scheduled for the period from June 4 through 15). ME6 has a deliberately short insertion of a COMBI 40+ implant with preserved residual (low frequency) hearing in the implanted cochlea. She was referred to us by our colleagues in Frankfurt, Germany. The studies with her will include measures of forward masking, with electrical stimuli as the masker and acoustic stimuli as the probe, and *vice versa*, to evaluate possible interactions between the two types of stimuli. In addition, the studies will include evaluation of additional speech processor designs, using combined electric and acoustic stimulation of the same cochlea. We expect that some of those designs will be informed by the results of the forward masking measures, *e.g.*, we may be able to combine electric and acoustic stimuli in ways that will minimize interference between the two.
- A visit by Reinhold Schatzer to New Fairfield, CT, to work with consultant Marian Zerbi in the further development of monitor programs for specification and control of psychophysical studies with recipients of bilateral cochlear implants (June 7-9).
- Studies with Ineraid subject SR3, in a continuing series of return visits by this subject (present studies are scheduled for the period from June 18 through 29). She is a recipient of the Ineraid device. The present studies with her will include an extensive evaluation of "conditioner pulses" processors, with many levels for the conditioner pulses and with different methods for presenting the conditioner pulses in conjunction with "data pulses" produced by the channel outputs of the speech processor. The present studies also will include (1) recordings of intracochlear evoked potentials for single polarities of stimulation and "split phase," monophasic-like pulses (but with full balancing of charge) and (2) additional measures for evaluation of effects of changes in values for various parameters of CIS processors, including changes in pulse rate, the cutoff frequency of the lowpass filters in the envelope detectors, and the exponent used in the mapping functions.
- A visit by consultant Chris van den Honert, for participation in evoked potential studies with subject SR3 and for advice on further development of the evoked potentials laboratory (June 26-28).
- Continued analysis of psychophysical, speech reception, and evoked potential data from current and prior studies.
- Continued preparation of manuscripts for publication.

VII. Acknowledgments

We thank subjects ME8 and ME9 for their participation in the studies of this quarter. We also thank Martin O'Driscol for his contributions to the studies with subject ME8.

.

Appendix 1: MATLAB Scripts for Processing Inputs Using Head-Related Transfer Functions

The scripts include two main (custom) functions and two helper functions. Brief descriptions of these functions are presented in the Tables below. Full descriptions of the main functions are presented in section III of this report. Listings for all of the scripts are presented in the remainder of this Appendix.

Main Functions

DoHRTF.m	Convolves input wave (*.wav) file with MIT KEMAR HRTFs for a given azimuth angle and a given elevation angle (page 30)
Mixer.m	Mixes HRTF-processed speech signals with HRTF-weighted noise (page 34)

Helper Functions

readhrtf.m	Public domain script provided by the MIT Media Laboratory, to read their HRTF data files (page 38)
hrtfpath.m	Public domain script provided by the MIT Media Laboratory, and is called in the execution of readhrtf.m (page 39)

```
function dohrtf(gain,azim,elev,select,directory,side,structdir)
% function DoHRTF([gain[,azim[,elev[,select[,directory[,side[,structdir]]]]]]])
%
% Processes wave files with MIT KEMAR HRTFs.
%
%  gain        Overall scaling factor < 1, at which processing shall start.
%              Default is 1.0.
%  azim        Azimuth in degrees. 0 is front, 90 is right, 270 is left. Can be a scalar or vector.
%              Default is 0 degrees (frontal incidence).
%  elev        Elevation in degrees. Can be a scalar or vector.
%              Default is 0 degrees elevation (median plane).
%  select      'L' use full data from left pinna (not supported yet)
%              'R' use full data from right pinna (not supported yet)
%              'C' use compact data (equalized to compensate for speaker response)
%              'D' use diffuse-field equalized data (compensated for speaker and ear canal response)
%              Default is 'D'
%  directory   Directory containing wave files to process.
%  side        Side which is HRTF processed with stereo input wave files ('left' or 'right').
%              Default is left side.
%  structdir   If structdir ~= 0, processed wave files will be saved in subdirectories, sorted by
%              original input file name and elevation (e.g. file\elev0\ for file.wav at elevation 0).
%              If structdir == 0, processed wave files will all be stored in a subdirectory 'hrtf'
%              Default is structdir = 0
%
% Reinhold Schatzer, 2000,2001 RTI

% History:
%
% Version 1.3, 02/21/01
%    - Bugfix in clipping check: absolute values of result vectors are checked for being >= 1 - 2^-15
%      (maximum in 16 bit signed fractional format) instead for being >= 1
%
% Version 1.2, 01/23/01
%    - Argument 'side' added. If not provided, dialog box is popped up to
%      enter side which shall be HRTF processed with stereo input wave files.
%    - Argument 'directory' moved to 5th position in argument list
%
% Version 1.1, 11/08/00
%    - '\n' replaced with '\r\n' in fprintf() to logfile.txt
%
% Version 1.0, 08/17/00
%    - First running version: processed CCITT noise file for elev = 0, azim = -90, 0, 90
%                                       NU6, VCV and CV signal files for elev = 0, azim = 0
%    - Fixed save gain to save only variable gain, not whole work space


% Epsilon to add to maxout for gain compensation. With 16 bit signed samples, eps
% is 2^-15, let's use 2 times this value.
eps = 2^-14;

% Start value for gain factor applied to convolution result. If after convolution with HRTFs signal
% magnitudes > 1 are encountered (which would be clipped to 1 in wavwrite), processing of all wave
% files is resumed with gain = 1/maxout
gainok = 0;                    % set to 1 at end if gain was ok to exit while loop
restart = 0;        % set to 1 if output magnitudes > 1 are encountered -> restart processing with new gain
sidestring   = {'Left' 'Right'}; % cell of strings for messages (array of strings need same number of characters)

% Set to 0 avoid plotting of convolved and convolved/cropped signals
doplots = 0;

%
% Check input arguments
%
if (nargin < 7)
   structdir = 0;
end
if (nargin < 6)
   side = 0;      % ask for side later
else
   if strcmp( upper(side), 'RIGHT' )
      sideidx = 2;   % right side signal is in column 2 of matrix returned by wavread()
   else
      sideidx = 1;   % left side signal is in column 1 of matrix returned by wavread()
   end
end
if (nargin < 5)
   [filename, directory] = uigetfile('*.wav', 'Please select a wave file to process all files in that directory!');
   if ~filename     % user pressed cancel
      return
   end
end
if (nargin < 4)
   select = 'D';
end
if (nargin < 3)
   elev = 0;
end
if (nargin < 2)
   azim = 0;
end
if (nargin < 1)
   gain = 1;
end

% Convert negative azimuths for left incidence [-1, -180] to positive ones [359, 180]
```

```
azim = mod( azim + 360, 360 );

select = upper(select);
if ( select ~= 'L' & select ~= 'R' & select ~= 'C' & select ~= 'D' )
   error( 'Wrong data set character provided for argument <select>.' );
end
if ( select == 'L' | select == 'R' )
   error( 'Processing with full left/right HRTF data not supported yet.' );
end

% Elevations and azimuth stepsizes at which data are provided
elevs = [-40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90;
   56 60 72 72 72 72 72 60 56 45 36 24 12 1];

for i = 1:length(elev)
          idx = find( elevs(1,:) == elev(i) );
          if isempty(idx)
          error( sprintf( 'No HRTF data for elevation at %d degrees available.', elev ) );
   elseif (length(idx) > 1)
      error( 'Elevation angle specified multiple times in input argument elev.' );
   end
          if ( any( mod( azim, 360/elevs(2,idx) ) ) & idx ~= 14 )
          error( sprintf( 'No HRTF data for provided azimuth angle(s) at elevation of %d degrees available.', elev ) );
          end
end

%
% Process all wave files in selected directory with selected HRTFs
%

%
% Create hrtf subdirectory to save the processed wave files in
%
if (~structdir)
          outdir = [ directory filesep 'hrtf' ];
   [status, result] = dos( [ 'md ' outdir ] );
end

% Open log file
logid = fopen( [ outdir filesep 'logfile.txt' ], 'w' );
if (logid == -1)
   error( sprintf( 'Unable to open log file %s.', [ directory filesep 'logfile.txt' ] ) );
end

% Iteratively loop through processing of all wave files until no output magnitude > 1 is encountered
while ~gainok

% Get structure array of root directory content
dirlist = dir(directory);

for i = 1:length(dirlist)
   len = length(dirlist(i).name);
   if ( ~dirlist(i).isdir & ( lower( dirlist(i).name(len-3:len) ) == '.wav' ) )

          %
             % Print/log process info
      %
      disp( sprintf( 'Processing wave file %s...', dirlist(i).name ) );
      fprintf( logid, 'Processing wave file %s...\r\n', dirlist(i).name );

      %
      % Open wave file and check sampling rate and sample bit length
      %
      [input,fs,nbits] = wavread( [directory filesep dirlist(i).name] );
      if (mod(44100,fs) ~= 0) % sampling rate not 44.1 kHz or integer fractional of that
          warning( sprintf( 'Skipping wave file %s, since sampling rate is not 44.1 kHz or a fractional of that.', dirlist(i).name ) );
         fprintf( logid, 'Skipping wave file %s, since sampling rate is not 44.1 kHz or a fractional of that.\r\n', dirlist(i).name );
      elseif (nbits ~= 16)
         warning( sprintf( 'Skipping wave file %s, since samples are not 16 bits long.', dirlist(i).name ) );
          fprintf( logid, 'Skipping wave file %s, since samples are not 16 bits long.\r\n', dirlist(i).name );
      else

         %
                % Check for stereo data in input wave file
          %
                if (size(input,2) > 1)
            if ~side    % true if side not provided as function argument
               button = questdlg('Wave file directory contains stereo input file(s). Which side shall be processed with HRTF functions?',...
                                 'Side selection','Left','Right','Left');
               if strcmp(button,'Right')
                  sideidx = 2;
               else
                  sideidx = 1;
               end
               side = 1;   % process selected side with all stereo input files
               disp( sprintf( '   Selected %s side as channel to be processed with stereo input wave files.', char(sidestring(sideidx)) ) );
               fprintf( logid, '   Selected %s side as channel to be processed with stereo input wave files.\r\n', char(sidestring(sideidx))
);
            end
            input = input(:,sideidx);   % process only data for selected side
            disp( sprintf( '   File %s contains stereo data. %s channel is convolved with HRTFs.', dirlist(i).name, char(sidestring(sideidx))
) );
            fprintf( logid, '   File %s contains stereo data. %s channel is convolved with HRTFs.\r\n', dirlist(i).name,
char(sidestring(sideidx)) );
         end
```

```
%
% Resample wave file data if fs ~= 44.1 kHz (we have integer fractional of that, then)
%
if (fs ~= 44100)
   disp( sprintf( 'Data in file %s had to be resampled at 44.1 kHz.', dirlist(i).name ) );
   fprintf( logid, 'Data in file %s had to be resampled at 44.1 kHz.', dirlist(i).name );
   input = interp( input, 44100/fs );
end

      %
 % Create subdirectory with the input wave file's name
 %
if (structdir)
 outdirbase = [ directory filesep dirlist(i).name(1:len-4) ];
          [status, result] = dos( [ 'md ' outdirbase ] );
end

 %
    % Get HRTF data, convolve and store result to stereo output wave file
       %
 for j = 1 : length(elev)
       %
          % Create subdirectory elevN for output files at elevation N
 %
   if (structdir)
          outdir = sprintf( '%s%selev%d', outdirbase, filesep, elev(j) );
                      [status, result] = dos( [ 'md ' outdir ] );
   end
   for k = 1 : length(azim)
          %
    % Processing with compact or diffuse-field HRTFs. They are symmetrical for left/right
       % incidence, so only HRTFs for azimuths 0 <= azim <= 180 are provided.
          % PROCESSING WITH FULL HRTFs STILL TO IMPLEMENT
             %

    %
       % Convolve input with left and right HRTFs at (elev,azim)
          %
             if (azim(k) > 180)
       % Read right HRTF at 360-azim for left incidence and switch sides when writing wave file.
          % ReadHRTF() returns left HRTF in column 1, right in column 2
             hrtf = readhrtf( elev(j), 360-azim(k), select );
             else
              hrtf = readhrtf( elev(j), azim(k), select );
    end

      % Length of convolving result is length(input)+length(hrtf)-1
          leftout  = conv( input(:,1), hrtf(:,1) );
                        rightout = conv( input(:,1), hrtf(:,2) );
      if (doplots)
         fvec = 1:size(leftout,1);
         fvec = 1000/44100 * fvec;       % frequency vector in ms
         figure(1); subplot(2,1,1); plot(leftout);
         title( sprintf( 'Left HRTF convolved output of file %s', dirlist(i).name ) );
         xlabel( 'Time [ms]');
         ylabel( 'Normalized amplitude');
         figure(2); subplot(2,1,1); plot(rightout);
         title( sprintf( 'Right HRTF convolved output of file %s', dirlist(i).name ) );
         xlabel( 'Time [ms]');
         ylabel( 'Normalized amplitude');
      end

      % Discard first/last length(hrtf)/2 = 64 samples of result vector
      leftout  = gain .* leftout( size(hrtf,1)/2 + 1 : size(input,1) + size(hrtf,1)/2 );
       rightout = gain .* rightout( size(hrtf,1)/2 + 1 : size(input,1) + size(hrtf,1)/2 );
      if (doplots)
         fvec = 1:size(leftout,1);
         fvec = 1000/44100 * fvec;       % frequency vector in ms
         figure(1); subplot(2,1,2); plot(leftout);
         title( sprintf( 'Left HRTF convolved and cropped output of file %s', dirlist(i).name ) );
         xlabel( 'Time [ms]');
         ylabel( 'Normalized amplitude');
         figure(2); subplot(2,1,2); plot(rightout);
         title( sprintf( 'Right HRTF convolved and cropped output of file %s', dirlist(i).name ) );
         xlabel( 'Time [ms]');
         ylabel( 'Normalized amplitude');
           disp( 'Press any key to continue...' ); pause;
      end

      maxout = max( max( abs([leftout rightout]) ) );
      if ( maxout >= 1 - 2^-15 )       % check if samples are out of 16 bit signed fractional range
         gain = gain / (maxout+eps);     % compensate gain for next iteration
         disp( '-------------------------------------------------------------------------------------------' );
         warning( sprintf( 'Samples out of range ]-1,+1[! Repeating HRTF processing with gain %5.3f.', gain ) );
         disp( '-------------------------------------------------------------------------------------------' );
         fprintf( logid, '-------------------------------------------------------------------------------------------\r\n' );
         fprintf( logid, 'Samples out of range ]-1,+1[! Repeating HRTF processing with gain %5.3f.\r\n', gain );
         fprintf( logid, '-------------------------------------------------------------------------------------------\r\n' );
         restart = 1;
         break
      end

      % Compose output file name and path and write stereo output wave file
      outfile = sprintf( '%s%c%s_%de%03da.wav', outdir, filesep, dirlist(i).name(1:len-4), elev(j), azim(k) );
```

```
               if (azim(k) > 180)
                  % Switch left and right channel to achieve left sound incidence.
                  % Column 1 of samples matrix is left channel, column 2 right channel.
                      wavwrite( [ rightout leftout ], 44100, 16, outfile );
                      else
                       wavwrite( [ leftout rightout ], 44100, 16, outfile );
                                 end

               disp( sprintf( '   elev %d azim %d done', elev(j), azim(k) ) );
               fprintf( logid, '   elev %d azim %d done\r\n', elev(j), azim(k) );

            end     % for k = 1 : length(azim)
            if restart        % exit j-loop
               break
            end
         end        % for j = 1 : length(elev)
         if restart % exit i-loop
            break
         end
      end % if (fs ~= 44100)
   end    % end of if() checking that we have .wav file
end       % for i = 1:length(dirlist)

if ~restart
   gainok = 1;
else
   restart = 0; % clear restart flag for next run
end

end       % while ~gainok

% Save and write out final gain factor and close log file
curdir = cd;
cd(outdir);
save gain gain;
cd(curdir);
disp( '------------------------------------------------------------------------------------------------' );
disp( sprintf( ' Processing completed on %s', datestr(now) ) );
disp( sprintf( ' Final gain factor used: %12.10f', gain ) );
disp( '------------------------------------------------------------------------------------------------' );
fprintf( logid, '\r\n------------------------------------------------------------------------------------------------\r\n' );
fprintf( logid, ' Processing completed on %s\r\n', datestr(now) );
fprintf( logid, ' Final gain factor used: %12.10f\r\n', gain );
fprintf( logid, '------------------------------------------------------------------------------------------------\r\n' );
fclose(logid);
```

```
function mixer(snr,gain,sigdir,noisefile,outdir,noiseazim,cal)
% function Mixer([snr[,gain[,sigdir[,noisefile[,outdir[,noiseazim[,cal]]]]]]])
%
% Mixes HRTF processed speech signals with HRTF weighted noise. Processes also speech signals
% in quiet by adjusting gain to overall gain used in adding noise.
%
% Parameters:
%  snr          Desired signal to noise ratio.
%               Default SNR is 0 dB
%  gain         Overall gain, at which iterative signal/noise mixing procedure shall start. This iterative
%               procedure finishes, when all samples are in the no-clipping range ]-1,+1[.
%               Default for overall gain is 1
%  sigdir       Directory containing HRTF processed signal wave files.
%  noisefile    Complete name and path of 1 noise wave file to be mixed. The noise files corresponding to the
%               specified azimuths, which are found in the same directory, will be processed (noise wave file
%               names are expected to end with '_XeYYYa.wav' for elevation X, azimuth YYY.
%  outdir       Output directory where mixed wave files will be saved.
%               Default is subdirectory 'Mixed_-10dB' for SNR = -10dB, etc.
%  noiseazim    Azimuth in degrees. 0 is front, 90 is right, 270 is left. Can be a scalar or vector.
%               Default is [270 0 90] degrees (left, front and right incidence).
%  cal          Calibration factor (signal/noise ratio), derived experimentally from signal/noise calibration
%               measurements.
%               Default is 16446/15392, which is factor for medial consonant tokens (vcvcal.wav) and
%               CCITT noise (ccittcal.wav).
%
% Reinhold Schatzer, 2000,2001 RTI

% History:
%
% Version 1.4, 03/08/01
%    - Added ramp at on- and offset of noise. Parameter tramp below can be set to 0 if noise
%      on/offset shall not be ramped, but start/stop at full amplitude.
%
% Version 1.3, 02/21/01
%    - Noise file is now resampled at 44.1 kHz too (like signal wave file), if sampling rate
%      is fractional of that
%    - Bugfix in clipping check: absolute values of result vectors are checked for being >= 1 - 2^-15
%      (maximum in 16 bit signed fractional format) instead for being >= 1
%
% Version 1.2, 01/24/01
%    - Indexing bug for filling in noise in output signal matrix (when signal is longer than noise) fixed
%
% Version 1.1, 11/08/00
%    - If SNR is not provided in function call, input dialog is popped up with default 0 dB
%    - '\n' replaced with '\r\n' in fprintf() to logfile.txt
%
% Version 1.0, 08/17/00
%    - First running version: mixed HRTF processed signal files NU6, VCV and CV (frontal incidence) with
%                             HRTF processed CCITT noise from left, front and right at
%                             SNR = +0dB, +5dB, +10dB and +15dB
%    - Fixed save gain to save only variable gain, not whole work space


% Set to 0 avoid plotting of convolved and convolved/cropped signals
doplots = 0;

% Parameters for onset/offset ramp of leading/trailing noise
fsample = 44100;  % sampling rate of signal in Hz
tramp = 1500;     % duration of onset/offset ramp in msecs (assume fsample = 44100 Hz)
tsustain = 300;   % duration of noise at full amplitude before/after test signal
% Smooth ramp scaling vector from 0 to 99.995% (values in range [0,1[) over tramp msecs at fsample Hz
% (50% amplitude reached after 0.3455*tramp msecs, 90% after 0.5584*tramp msecs)
rampgain = 1 - exp( -( 2.5.*linspace(0,1,fsample*tramp/1000) ) .^ 2.5 )';

% Epsilon to add to maxout for gain compensation. With 16 bit signed samples, eps
% is 2^-15, let's use 2 times this value. The bigger this value, the faster
% the iterative process yielding to non-clipped wave files will be.
eps = 2^-14;

% Init values for iterative signal mixing procedure. If after mixing of signal and noise waves
% magnitudes >= 1 are encountered (which would be clipped wavwrite()), processing of all wave
% files is resumed with gain = 1/(maxout+eps)
gainok = 0;         % set to 1 at end if gain was ok to exit while loop
remix = 0;          % set to 1 if output magnitudes > 1 are encountered -> restart processing with new gain
gotoutdir = 1;

%
% Check input arguments
%
if (nargin < 7)
   cal = 16446/15392;         % cal > 1 if (cal signal amp) > (cal noise amp)
end
if (nargin < 6)
          noiseazim = [270 0 90];
end
azimchar = 'LFR';
if (nargin < 5)
   gotoutdir = 0;
end
if (nargin < 4)
   [noisefile, path] = uigetfile( '*.wav', 'Select 1 wave file in directory containing NOISE files to be mixed...' );
   if ~noisefile    % user pressed cancel
      return
   end
   % Compose full noise file name (path ends with '\')
```

```
   noisefile = [ path noisefile ];
end
if (nargin < 3)
   [filename, sigdir] = uigetfile('*.wav', 'Select 1 SIGNAL wave file to mix all files in that directory...');
   if ~filename     % user pressed cancel
      return
   end
else
   sigdir = [ sigdir filesep ];         % Add filesep to user provided directory with signal files
end
if (nargin < 2)
   gain = 1;
end
if (nargin < 1)     % Ask for SNR if it has not provided as argument
   snr = inputdlg( {'SNR [dB]'}, 'Please enter SNR in dB...', 1, {'0'} );
   if isempty(snr)
      return                  % Quit, if user clicked Cancel
   else
      snr = str2num(snr{1});
   end
end

% Compose path name of output directory if it has not been specified by user
if ~gotoutdir
          if tramp > 0
      outdir = sprintf( '%sMixed_ramp_%+ddB', sigdir, snr );
   else
      outdir = sprintf( '%sMixed_%+ddB', sigdir, snr );
   end
end
% Create output directory
[status, result] = dos( [ 'md ' outdir ] );


% Calculate gain factor for signal and noise from given SNR, considering
% calibration SNR ( cal > 1 if (cal signal amp) > (cal noise amp) )
signalgain = 1;
noisegain  = cal / 10^(snr/20);

% Convert negative azimuths for left incidence [-1, -180] to positive ones [359, 180]
noiseazim = mod( noiseazim + 360, 360 );

% Remove trailing '_XeYYYa.wav' characters from full noise filename
noisefilebase = noisefile(1:length(noisefile)-11);

% Open log file
logid = fopen( [ outdir filesep 'logfile.txt' ], 'w' );
if (logid == -1)
   error( sprintf( 'Unable to open log file %s.', [ sigdir filesep 'logfile.txt' ] ) );
end


%
% Iteratively loop through mixing signal with noise until no output magnitude > 1 is encountered
%

dirlist = dir(sigdir);        % get contents structure array of directory with signal wave files

while ~gainok

for n = 1:length(noiseazim)

          %
          % Open noise wave file for each azimuth
          %
   noisefile = sprintf( '%s_0e%03da.wav', noisefilebase, noiseazim(n) );
   [noise,fs,nbits] = wavread(noisefile);
   if (size(noise,2)) ~= 2
         warning( sprintf( 'Skipping mono noise wave file %s, since stereo wave file is needed.', noisefile ) );
      fprintf( logid, 'Skipping mono noise wave file %s, since stereo wave file is needed.\r\n', noisefile );
   elseif (mod(44100,fs) ~= 0)          % sampling rate not 44.1 kHz or integer fractional of that
      warning( sprintf( 'Skipping noise wave file %s, since sampling rate is not 44.1 kHz or fractional of that.', noisefile ) );
      fprintf( logid, 'Skipping noise wave file %s, since sampling rate is not 44.1 kHz or fractional of that.\r\n', noisefile );
   elseif (nbits ~= 16)
      warning( sprintf( 'Skipping noise wave file %s, since samples are not 16 bits long.', noisefile ) );
      fprintf( logid, 'Skipping noise wave file %s, since samples are not 16 bits long.\r\n', noisefile );
   else

            %
      % Resample noise file data if fs ~= 44.1 kHz (we have integer fractional of that, then)
            %
      if (fs ~= 44100)
               disp( sprintf( '  Data in file %s had to be resampled at 44.1 kHz.', noisefile ) );
                   fprintf( logid, '  Data in file %s had to be resampled at 44.1 kHz.\r\n', noisefile );
         noise(:,1) = interp( noise(:,1), 44100/fs );       % left channel
         noise(:,2) = interp( noise(:,2), 44100/fs );       % right channel
             end

             %
          % Print process info
                   %
                   disp( sprintf( 'Adding noise from file %s... (azim = %d)', noisefile, noiseazim(n) ) );
      fprintf( logid, 'Adding noise from file %s... (azim = %d)\r\n', noisefile, noiseazim(n) );

      for i = 1:length(dirlist)
```

```
                    len = length(dirlist(i).name);
                    if ( ~dirlist(i).isdir & ( lower( dirlist(i).name(len-3:len) ) == '.wav' ) )

                    %
                    % Open signal wave file and check sampling rate and sample bit length
                       %
                    [signal,fs,nbits] = wavread( [sigdir filesep dirlist(i).name] );
               if (size(signal,2)) ~= 2
                  warning( sprintf( 'Skipping mono signal wave file %s, since stereo wave file is needed.', dirlist(i).name ) );
                  fprintf( logid, '  Skipping mono signal wave file %s, since stereo wave file is needed.\r\n', dirlist(i).name );
               elseif (mod(44100,fs) ~= 0) % sampling rate not 44.1 kHz or integer fractional of that
                  warning( sprintf( 'Skipping signal wave file %s, since sampling rate is not 44.1 kHz or fractional of that.', dirlist(i).name
) );
                  fprintf( logid, '  Skipping signal wave file %s, since sampling rate is not 44.1 kHz or fractional of that.\r\n',
dirlist(i).name );
                        elseif (nbits ~= 16)
                  warning( sprintf( 'Skipping signal wave file %s, since samples are not 16 bits long.', dirlist(i).name ) );
                  fprintf( logid, '   Skipping signal wave file %s, since samples are not 16 bits long.\r\n', dirlist(i).name );
                    else

                       %
                            % Resample wave file data if fs ~= 44.1 kHz (we have integer fractional of that, then)
                       %
                            if (fs ~= 44100)
                          disp( sprintf( '  Data in file %s had to be resampled at 44.1 kHz.', dirlist(i).name ) );
                              fprintf( logid, '  Data in file %s had to be resampled at 44.1 kHz.\r\n', dirlist(i).name );
                                signal(:,1) = interp( signal(:,1), 44100/fs );  % left channel
                                signal(:,2) = interp( signal(:,2), 44100/fs );  % right channel
                       end

                                %
                            % Print process info
                            %
                  disp( sprintf( '  Mixing signal wave file %s with noise from azim = %d.', dirlist(i).name, noiseazim(n) ) );
                  fprintf( logid, '  Mixing signal wave file %s with noise from azim = %d.\r\n', dirlist(i).name, noiseazim(n) );

                  %
                  % Add signal to noise by maintaining tramp+tsustain msecs of leading and trailing noise
                  % (noise is looped, if needed).
                  %

                  % Reserve space for output sample matrix and fill in with SNR scaled noise
                  output = zeros( size(signal,1) + 2*fsample*(tramp+tsustain)/1000, 2 );     % tramp+tsustain msecs heading/trailing noise
                  for j = 1:floor( size(output,1) / size(noise,1) )
                       output( (j-1)*size(noise,1)+1:j*size(noise,1), : ) = noisegain .* noise;
                  end
                  if isempty(j)
                     j = 0;      % set j to 0 in case for loop has not been entered, i.e. noise longer than output
                  end
                  output( j*size(noise,1)+1:size(output,1), : ) = noisegain .* noise( 1:size(output,1)-j*size(noise,1), : );

                  % Ramp noise if tramp > 0
                  if tramp > 0
                     ovrlrampgain = ones(size(output,1),1);       % fill in scaling factors 1 first for length of output
                     ovrlrampgain(1:length(rampgain)) = rampgain; % overwrite with startup ramp noise scaling factors
                     ovrlrampgain(size(output,1)-length(rampgain)+1:size(output,1)) = ...
                        rampgain(length(rampgain):-1:1);          % overwrite with mirrored ramp noise scaling factors at end
                     output(:,1) = ovrlrampgain .* output(:,1);   % ramp left side noise
                     output(:,2) = ovrlrampgain .* output(:,2);   % ramp right side noise
                  end

                  % Add signal after tramp+tsustain msecs and scale result with overall gain
                  output( fsample*(tramp+tsustain)/1000+1 : size(output,1)-fsample*(tramp+tsustain)/1000, : ) = ...
                     output( fsample*(tramp+tsustain)/1000+1 : size(output,1)-fsample*(tramp+tsustain)/1000, : ) + signalgain .* signal;
                  output = gain .* output;

                  if (doplots)
                     fvec = 1:size(output,1);
                     fvec = 1000/44100 * fvec;       % frequency vector in ms
                     figure(1); subplot(2,1,1); plot(fvec,output(:,1));
                     title( 'Left side output of mixed signal' );
                     xlabel( 'Time [ms]');
                     ylabel( 'Amplitude');
                     subplot(2,1,2); plot(fvec,output(:,2));
                     title( 'Right side output of mixed signal' );
                     xlabel( 'Time [ms]');
                     ylabel( 'Amplitude');
                  end

                  maxout = max( max( abs(output) ) );
                  if ( maxout >= 1 - 2^-15 )       % check if samples are out of 16 bit signed fractional range
                     gain = gain / (maxout+eps);     % compensate gain for next iteration
                     disp( '-------------------------------------------------------------------------------------------------' );
                     warning( sprintf( 'Samples out of range ]-1,+1[! Mixing again with gain %5.3f.', gain ) );
                     disp( '-------------------------------------------------------------------------------------------------' );
                     fprintf( logid, '-------------------------------------------------------------------------------------------------\r\n' );
                     fprintf( logid, 'Samples out of range ]-1,+1[! Mixing again with gain %5.3f.\r\n', gain );
                     fprintf( logid, '-------------------------------------------------------------------------------------------------\r\n' );
                     remix = 1;
                     break
                  end

                  % Compose output file name acc. to CxxsN.wav (C = M,F,L - xx = 01..24 - s = 1..3 - N = 0,L,F,R)
                  % and write stereo output wave file
                  outfile = [ outdir filesep sprintf( '%s%c.wav', dirlist(i).name(1:len-11), azimchar(n) ) ];
```

```
                    wavwrite( output, 44100, 16, outfile );

                end       % if (size(signal,2)) ~= 2
                end       % end of if() checking that we have .wav file
                end       % for i = 1:length(dirlist)
         end       % if (size(noise,2)) ~= 2
   if remix        % exit n-loop
      break
   end
end       % for n = 1:length(noiseazim)

if ~remix
   gainok = 1;
else
   remix = 0; % clear remix flag for next run
end

end       % while ~gainok


%
% Gain-compensate front incidence signal wave files with gain factor finally used
% in mixing signal with noise from left, front and right.
% Put wave files in same directory as files with mixed noise
%

for i = 1:length(dirlist)
          len = length(dirlist(i).name);
   if ( ~dirlist(i).isdir & ( lower( dirlist(i).name(len-3:len) ) == '.wav' ) )

          %
             % Open signal wave file and check sampling rate and sample bit length
             %
          [signal,fs,nbits] = wavread( [sigdir filesep dirlist(i).name] );
                   if (mod(44100,fs) ~= 0)       % sampling rate not 44.1 kHz or integer fractional of that
          warning( sprintf( 'Skipping signal wave file %s, since sampling rate is not 44.1 kHz or fractional of that.', dirlist(i).name ) );
         fprintf( logid, 'Skipping signal wave file %s, since sampling rate is not 44.1 kHz or fractional of that.\r\n', dirlist(i).name );
                   elseif (nbits ~= 16)
         warning( sprintf( 'Skipping signal wave file %s, since samples are not 16 bits long.', dirlist(i).name ) );
         fprintf( logid, 'Skipping signal wave file %s, since samples are not 16 bits long.\r\n', dirlist(i).name );
      else

          %
                            % Resample wave file data if fs ~= 44.1 kHz (we have integer fractional of that, then)
             %
                            if (fs ~= 44100)
                   disp( sprintf( 'Data in file %s had to be resampled at 44.1 kHz.', dirlist(i).name ) );
                   fprintf( logid, 'Data in file %s had to be resampled at 44.1 kHz.\r\n', dirlist(i).name );
                               signal = interp( signal, 44100/fs );
          end

                      %
                            % Gain compensate and print process info
          %
         output = gain .* signal;
         disp( sprintf( 'Gain-compensating signal wave file in quiet %s.', dirlist(i).name ) );
             fprintf( logid, 'Gain-compensating signal wave file in quiet %s.\r\n', dirlist(i).name );

         % Compose output file name acc. to CxxsN.wav (C = M,F,L - xx = 01..24 - s = 1..3 - N = 0,L,F,R)
         % and write stereo output wave file
         outfile = [ outdir filesep sprintf( '%s0.wav', dirlist(i).name(1:len-11) ) ];
             wavwrite( output, 44100, 16, outfile );

          end       % if (fs ~= 44100)
          end       % end of if() checking that we have .wav file
end       % for i = 1:length(dirlist)

% Write out final gain factor and close log file
curdir = cd;
cd(outdir);
save gain gain;
cd(curdir);
disp( '-------------------------------------------------------------------------------------------' );
disp( sprintf( ' Processing completed on %s', datestr(now) ) );
disp( sprintf( ' Final gain factor used: %12.10f', gain ) );
disp( '-------------------------------------------------------------------------------------------' );
fprintf( logid, '\r\n-------------------------------------------------------------------------------------------\r\n' );
fprintf( logid, ' Processing completed on %s\r\n', datestr(now) );
fprintf( logid, ' Final gain factor used: %12.10f\r\n', gain );
fprintf( logid, '-------------------------------------------------------------------------------------------\r\n' );
fclose(logid);
```

```
function [y] = readhrtf(elev,azim,select)
%
% function [y] = readhrtf(elev,azim,select)
%
% elev is elevation from -40 to 90 degrees
% azim is azimuth from 0 to 180 degrees
% select is:
%         'L' use full data from left pinna
%         'R' use full data from right pinna
%         'C' use compact data (equalized to compensate for speaker response)
%         'D' use diffuse-field equalized data (compensated for speaker and ear canal response)
% Returns stereo symmetrical hrtf in first two columns of
% y such that left is first column, right is second column.
% Amplitude values are in the range [-1,+1].
%
% Bill Gardner
% Copyright 1995 MIT Media Lab. All rights reserved.
% Modified to read wave files by Reinhold Schatzer, 2000 RTI.

%
% Root directory containing HRTF data
%
root = 'c:\rtisys\matlab\hrtf\hrtfdata';

%
% check arguments
%
azim = round(azim);
if ((azim < 0) | (azim > 180))
          error('Azimuth must be between 0 and 180 degrees.');
end
if ((elev < -40) | (elev > 90))
          error('Elevation must be between -40 and 90 degrees.');
end


%
% format filename
%
flip_azim = 360 - azim;
if (flip_azim == 360)
          flip_azim = 0;
end
ext = '.wav';
if (select == 'L')
          pathname = hrtfpath(root,filesep,'full',select,ext,elev,azim);
   y = zeros(512,2);
   [y(:,1),fs,nbits] = wavread(pathname);
   if (fs ~= 44100 | nbits ~= 16)
      error('Incorrect wave file format. Expected 16 bit samples at 44.1 kHz sampling rate.');
   end
          pathname = hrtfpath(root,filesep,'full',select,ext,elev,flip_azim);
          [y(:,2),fs,nbits] = wavread(pathname);
   if (fs ~= 44100 | nbits ~= 16)
      error('Incorrect wave file format. Expected 16 bit samples at 44.1 kHz sampling rate.');
   end
elseif (select == 'R')
          pathname = hrtfpath(root,filesep,'full',select,ext,elev,flip_azim);
   y = zeros(512,2);
   [y(:,1),fs,nbits] = wavread(pathname);
   if (fs ~= 44100 | nbits ~= 16)
      error('Incorrect wave file format. Expected 16 bit samples at 44.1 kHz sampling rate.');
   end
          pathname = hrtfpath(root,filesep,'full',select,ext,elev,azim);
          [y(:,2),fs,nbits] = wavread(pathname);
   if (fs ~= 44100 | nbits ~= 16)
      error('Incorrect wave file format. Expected 16 bit samples at 44.1 kHz sampling rate.');
   end
elseif (select == 'C')
          pathname = hrtfpath(root,filesep,'compact','H',ext,elev,azim);
          [y,fs,nbits] = wavread(pathname);
   if (fs ~= 44100 | nbits ~= 16)
      error('Incorrect wave file format. Expected 16 bit samples at 44.1 kHz sampling rate.');
   end
elseif (select == 'D')
          pathname = hrtfpath(root,filesep,'diffuse','H',ext,elev,azim);
          [y,fs,nbits] = wavread(pathname);
   if (fs ~= 44100 | nbits ~= 16)
      error('Incorrect wave file format. Expected 16 bit samples at 44.1 kHz sampling rate.');
   end
else
          error(sprintf('%s not a valid selection, use L, R, C or D for L/R full, Compact or Diffuse data.',select));
end
```

```
function [s] = hrtfpath(root,dir_ch,subdir,select,ext,elev,azim)
%
% function [s] = hrtfpath(root,dir_ch,subdir,select,ext,elev,azim)
% Return pathanme for HRTF data file:
%         root is root directory.
%         dir_ch is directory character, '/' (unix) or ':' (mac).
%         subdir is 'compact', 'full', etc.
%         select is 'L', 'R' for L/R full data, 'H' for compact or diffuse data.
%         ext is the filename extension '.wav', etc.
%         elev is elevation.
%         azim is azimuth.
%
s = sprintf('%s%s%s%selev%d%s%s%de%03da%s',...
          root,dir_ch,subdir,dir_ch,round(elev),...
          dir_ch,select,round(elev),round(azim),ext);
```