
Open Source Software for Experiment Design and Control

TUTORIAL

James M. Hillenbrand
Western Michigan University,
Kalamazoo

Robert T. Gayvert
Gayvert Consulting,
Rochester, NY

The purpose of this paper is to describe a software package that can be used for performing such routine tasks as controlling listening experiments (e.g., simple labeling, discrimination, sentence intelligibility, and magnitude estimation), recording responses and response latencies, analyzing and plotting the results of those experiments, displaying instructions, and making scripted audio-recordings. The software runs under Windows and is controlled by creating text files that allow the experimenter to specify key features of the experiment such as the stimuli that are to be presented, the randomization scheme, interstimulus and intertrial intervals, the format of the output file, and the layout of response alternatives on the screen. Although the software was developed primarily with speech-perception and psychoacoustics research in mind, it has uses in other areas as well, such as written or auditory word recognition, written or auditory sentence processing, and visual perception.

KEY WORDS: experiment control software, speech perception software, stimulus presentation, open source software

This report describes a software package called Alvin, a general-purpose program for controlling stimulus presentation and the collection of participant responses for behavioral research. The program was developed primarily with speech-perception and psychoacoustics research in mind, although it has uses in other areas as well, such as written or auditory word recognition, written or auditory sentence processing, and visual perception. The program was named in honor of Alvin M. Liberman, the pioneering Haskins Laboratories speech researcher. The Alvin program has some features in common with E-Prime (MacWhinney, St. James, Shunn, Li, & Schneider, 2001), although it is structured rather differently. Unlike E-Prime, Alvin is made available at no charge in both executable and source form. This report will focus primarily on the use of Alvin to design experiments involving auditory stimuli. For those kinds of experiments, Alvin (a) presents auditory stimuli (monaural or stereo, any number per trial) in an order specified by the experimenter, (b) allows the specification of interstimulus and intertrial intervals, (c) records responses from participants using some combination of buttons, sliders, and text entry, and (d) records response latencies (but with a degree of accuracy that may not be adequate for all purposes; see below).

The software was developed under Microsoft Windows¹ using C++, the wxWidgets graphical user interface (GUI) library, and the *Tcl* scripting language. The C++ code provides a set of audio and GUI functions

¹Alvin should run on any Win32 system (Windows 98/ME/NT/2000/XP). Windows 3.1 is not supported.

that can be used from *Tcl*. The program plays and records with any Windows-compatible sound card.² The experiment control framework and the scripts used to design experiments are written in *Tcl*. The flexibility of *Tcl* allows new experiments to be written by users with limited programming knowledge. The software, which can be downloaded from <http://homepages.wmich.edu/~hillenbr>,³ includes a collection of control files and test stimuli for several sample experiments that were created with Alvin. The sample experiments, which probably represent the best source of documentation for the program, address topics such as phonetic labeling, speech discrimination (with and without feedback), open-set word identification, dichotic listening, speech intelligibility, prompted audio-recording, judgment of vocal quality, image perception, and the presentation of *mpeg* video.

Designing an experiment with Alvin requires, at minimum, the creation of two text files: (a) a *Tcl script* that controls features of the experiment such as the randomization scheme and the layout and behavior of but-

tons, sliders, and text-entry boxes on the screen and (b) a *stimulus-presentation file* that controls features such as the stimuli that are to be presented, interstimulus intervals, and (optionally) the nature of feedback to be presented to participants.

Sample Experiments

Before describing the format of the *Tcl* script and stimulus-presentation files, it might be useful to take a quick look at the screen layouts associated with a few Alvin-created experiments. Figure 1 shows the screen layout for a simple two-choice phonetic labeling experiment. On each trial the listener will hear a single consonant-vowel syllable drawn at random from a nine-step synthetic continuum ranging from /w/ to /r/. The listener's job is to click the appropriate button to indicate whether the syllable sounded more like *way* or *ray*. (If desired, Alvin provides a simple mechanism to allow the participant to respond by pressing a key on the computer keyboard in place of clicking a button, e.g., *w* for *way* and *r* for *ray*.) The bottom row of buttons includes *Start* and *Return to Main Menu*, which the participant uses to begin and end the session, respectively. Also included on the bottom row are three optional buttons: (a) a *Back Up* button that can be used to repeat a trial in which the participant inadvertently pressed the wrong button, (b) an *Instructions* button that will cause written instructions to be displayed on the screen, and (c) a *View Results* button that, in this case, will cause the labeling function to be displayed on the screen (this button can be disabled at the start of the experiment and enabled only when all trials have been completed). The experiment shown in Figure 1 is included in the folder *wrid* among the examples that are installed by the Alvin setup program.

Figure 2 shows another phonetic labeling experiment but with a larger number of response alternatives. The participant's job here is to press the appropriate

²The current version of Alvin uses the cross-platform *PortAudio* library for audio-recording and playback. *PortAudio* is implemented as a layer on top of native platform-specific audio services. Under Windows, *PortAudio* can be configured to use either Windows *Multimedia Extensions* or *DirectSound*. *PortAudio* supports a wide range of sampling rates (8–96 kHz) and sample formats (8–32 bits). During playback of a WAV file, Alvin configures *PortAudio* to match the characteristics of the audio file. *PortAudio* is intended to be compatible with any Windows sound card, but there is a wide range of audio hardware on the market, particularly for laptops, and Alvin has been tested on a rather small subset of the available devices. Although only lightly tested at present, the most recent release of Alvin has support for audio recording (but not playback) using Tucker & Davis Technologies audio hardware (www.tdt.com). (TDT-recorded signals are played using the Windows sound card.) See the folder *tdtexample*.

³The wxWidgets package itself is not distributed with the software and needs to be downloaded from <http://www.wxwidgets.org/>. In order to build Alvin, the environment variable WX needs to be associated with the directory name holding the wxWidgets library. The library is needed only if the user wishes to build Alvin from the source code.

Figure 1. Screen layout for a two-choice labeling experiment.

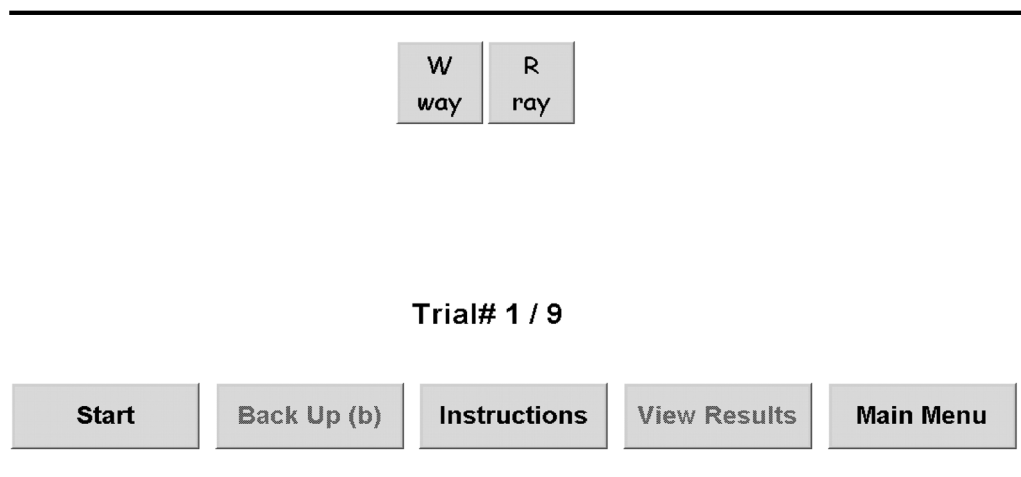
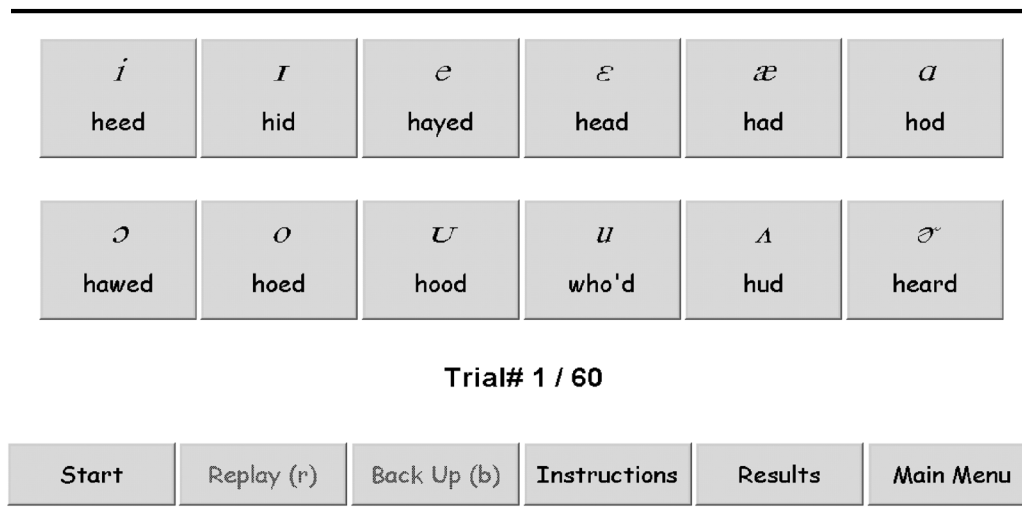


Figure 2. Screen layout for a vowel identification experiment.

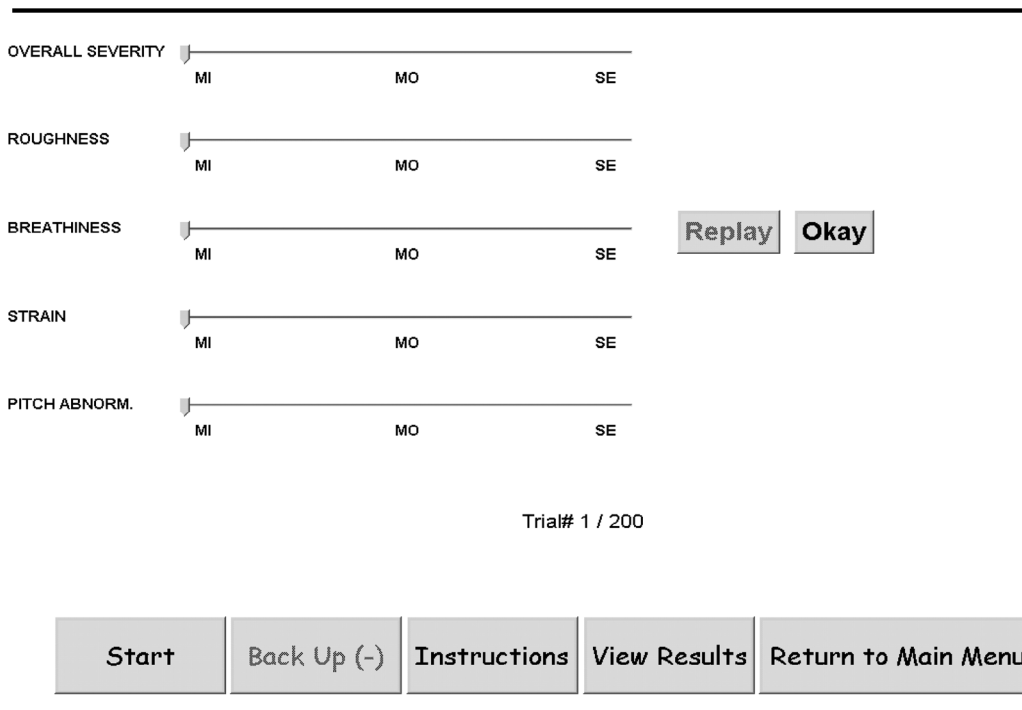


button to indicate which of 12 vowels was spoken. Note that this experiment includes an optional *Replay* button allowing the listener to hear the stimulus as many times as desired before entering a response. (Alvin keeps track of the number of times this button is used on each trial and records this information in the output file. We have found this “replay count” to be an indirect but sensitive measure of intelligibility in some of our work. A limit can be imposed on the number of times the *Replay* button is used—see the script *signalsinnoise.alvx* in the *signalsinnoise* folder for an example.) Note also that the

buttons are labeled with phonetic symbols. Displaying characters from fonts such as this requires special handling. This is described below. The experiment shown in Figure 2 is included in the folder *vowelid*.

Figure 3 shows the screen layout for an experiment asking listeners for judgments of voice quality for voice samples recorded from speakers with laryngeal disorders. A set of sliders is used to record participants’ ratings on five separate dimensions: (a) the overall severity of the voice quality disturbance, (b) roughness, (c) breathiness, (d) strain, and (e) pitch abnormality. The participant can

Figure 3. Screen layout for an experiment using multiple sliders.



press *Replay* as often as desired and is asked to click the *Okay* button when ratings have been provided on all five dimensions. The experiment shown in Figure 3 is included in the folder *grbs*. Figure 4 also focuses on vocal quality, but in this case a direct magnitude estimation technique is used. Listeners are asked to enter a number that is proportional to the degree of breathiness in the voice. The experiment shown in Figure 4 is included in the folder *br*.

Figure 5 shows an experiment in which participants are asked for two separate judgments on each trial. After hearing an utterance, listeners are asked to judge whether it was spoken by a boy or a girl. After making this judgment, listeners are asked to click on a number between 1 and 5 to express how much confidence they have in their judgment (a slider could just as easily be used for this purpose). The experiment shown in Figure 5 is included in the folder *bg* (see *bg3.tcl*; see also other approaches to the same problem in *bg.tcl*, and *bg2.tcl*).

Figure 6 shows Alvin being used to record speech samples from participants. The text string at the top of the screen prompts the talker for the word or sentence that is desired. An audio prompt may be used in addition to or in place of the text prompt. The display in the center of the screen is a peak-detecting record-level meter. The participant clicks the *Record* button when ready to speak, the *Playback* button to audition the recording, and the *Next* button to proceed to the next utterance. The experiment shown in Figure 6 is included in the folder *promptedrecording*.

Figure 7 shows an example involving images rather than auditory signals. Instructions for this experiment are as follows:

You will see a pair of pictures consisting of points arranged either horizontally or vertically. If the dots are arranged more-or-less horizontally, pick the figure (a or b) that is more nearly horizontal. If the dots are arranged more-or-less vertically, pick the figure (a or b) that is more nearly vertical.

This example is included to show that Alvin can be used in domains other than acoustic. The experiment shown in Figure 7 is included in the folder *imageexample*. Although not shown in any of the examples presented here, Alvin can also present stimuli consisting of text strings (e.g., words, phrases, or sentences). Examples using two quite different approaches to the presentation of text as stimuli can be found in the folders *textexample* and *mentalrotation*. An example involving the presentation of *mpeg* video can be found in the folder *videoexample*.

The Stimulus-Presentation File

A simple example of a stimulus-presentation file is shown below. This file was used for the experiment shown in Figure 1. The second column, and the only one that is mandatory, is simply a list of the sound files that will be played to the listener:

```
"1" wr01.wav  
"2" wr02.wav  
"3" wr03.wav  
"4" wr04.wav  
"5" wr05.wav  
"6" wr06.wav  
"7" wr07.wav
```

Figure 4. Screen layout for an experiment using direct magnitude estimation.

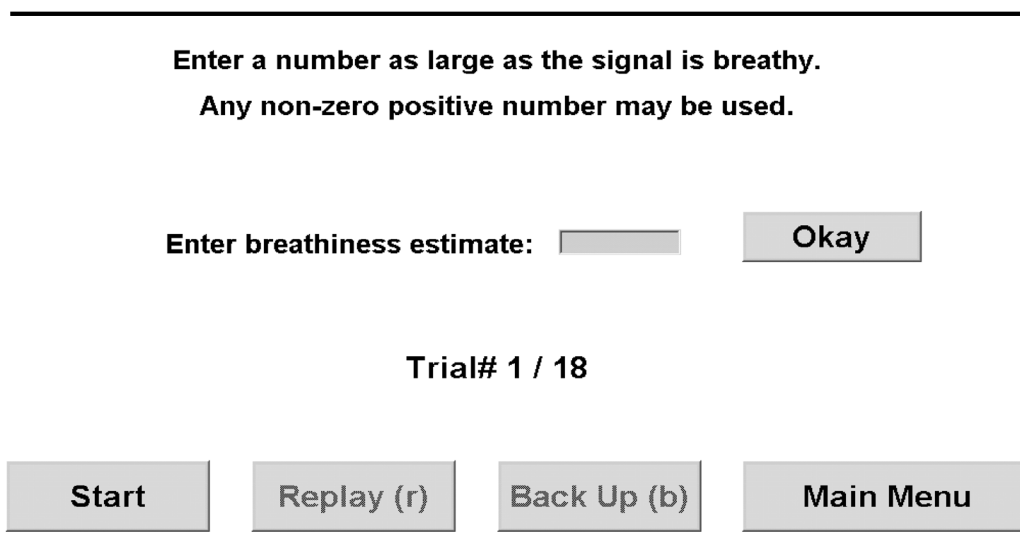
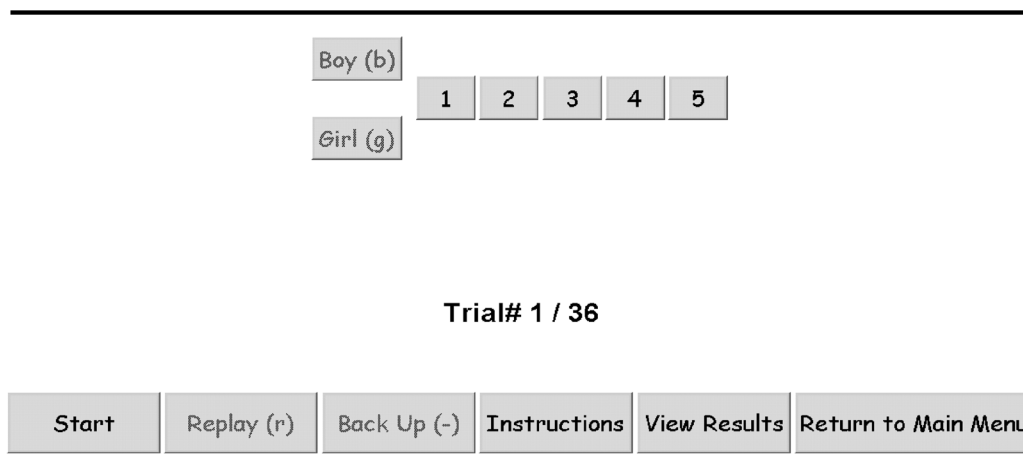


Figure 5. Screen layout for an experiment in which participants are asked for two judgments for each stimulus: (a) Was the utterance spoken by a boy or a girl? (b) What is your confidence in your judgment?



"8" wr08.wav
 "9" wr09.wav

A setting in the *Tcl* script will determine whether Alvin presents these stimuli in exactly the order in which they appear in the stimulus-presentation file or in random order. The first column, enclosed in double quotes, is an optional stimulus-description field. The stimulus-description field can contain any information that the experimenter might find useful when the data are analyzed—for example, codes specifying place of articulation, manner of articulation, and voicing, codes indicating whether the signal was spoken by a man or a woman, or codes indicating the signal-to-noise ratio, filter setting, or synthesis condition. This text will appear in the output file created by Alvin and can therefore be used to facilitate data analysis. In the present example, the stimulus-description field simply holds the stimulus number along the nine-point /w-/r/ continuum.

Shown below⁴ are a few lines from a stimulus-presentation file that would be used for an ABX discrimination experiment involving signals from the same nine-step /w-/r/ continuum:

```
"1 3 1 1" wr01.wav 500 wr03.wav 500
wr01.wav 1000 "Answer: 1"

"1 3 3 2" wr01.wav 500 wr03.wav 500
wr03.wav 1000 "Answer: 2"

"2 4 2 1" wr02.wav 500 wr04.wav 500
wr02.wav 1000 "Answer: 1"
```

Listeners hear three signals on each trial. The first two signals are always different, and the third signal always matches either stimulus 1 or stimulus 2. The

⁴The three sample entries that are shown in this example should each appear as a single line in the stimulus presentation file. Because of column-width limitations, the text occupies two lines in the manuscript. This applies to other examples in the article.

Figure 6. Screen layout for an audio-recording session.

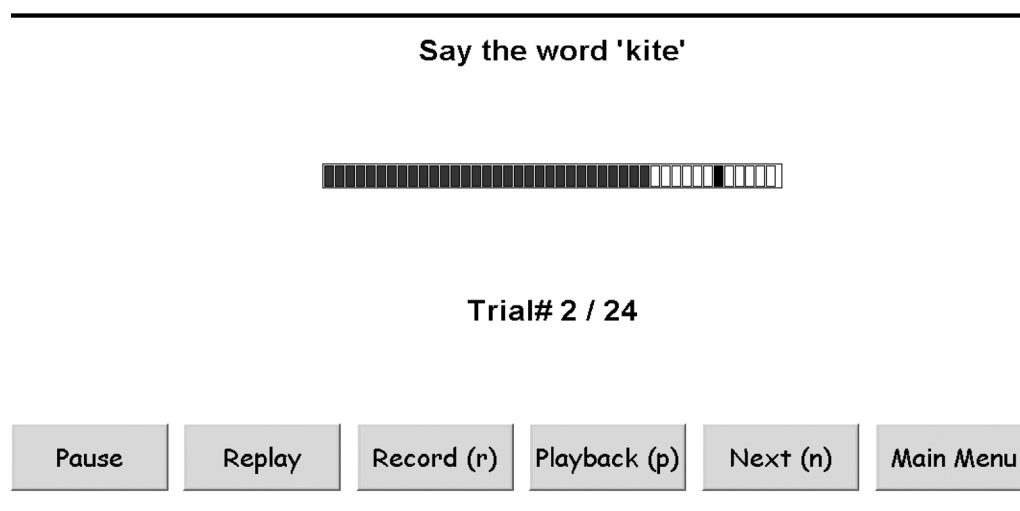
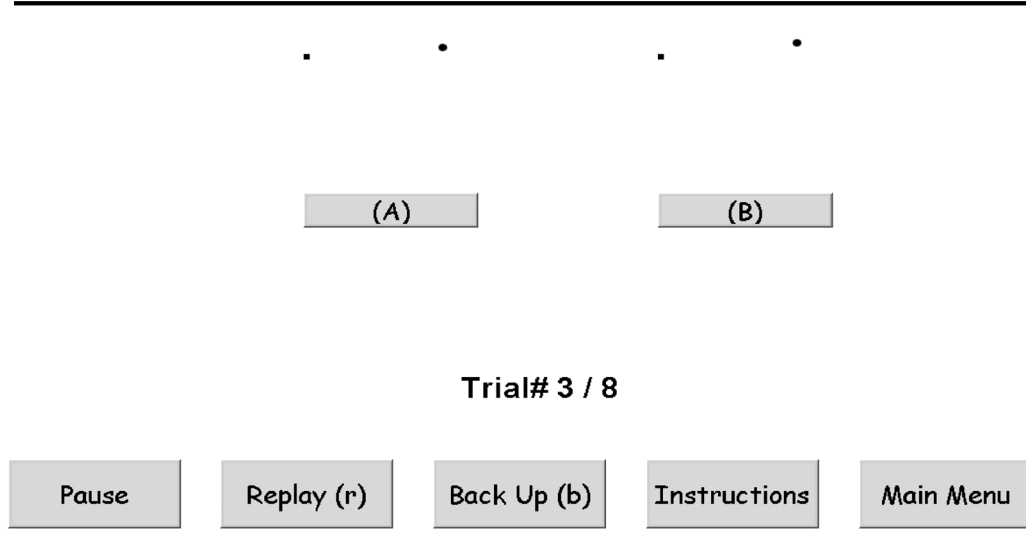


Figure 7. Screen layout for an experiment involving judgments based on images rather than sound.



participant is asked to judge whether the third stimulus matches the first or second stimulus. As before, the initial field in double quotes is an optional stimulus-description field (the stimulus numbers for the three stimuli followed by a stimulus number indicating the correct answer for the trial; e.g., a 1 will appear here if stimulus 3 matches stimulus 1). Taking the first line of this file as an example, the next six entries on this line will be interpreted by Alvin as follows: present *wr01.wav*, followed by a 500-ms delay; present *wr03.wav*, followed by a 500-ms delay; present *wr01.wav*; wait indefinitely for a response, then delay 1,000 ms before cycling to the next trial (this is the intertrial interval). There is no limit to the number of stimuli that can be presented on a given trial. Each filename is followed by an interstimulus interval in milliseconds, except for the last filename, which is followed by an intertrial interval. Although it is not done in this example, the number of stimuli, interstimulus intervals, and intertrial intervals can vary from one trial to the next. The last field in double quotes is an optional feedback field. The text that appears in this field will be displayed on the screen after the participant enters a response. For example, after responding to the trial specified in the first line of this file, the string "Answer: 1" will appear on the screen telling the listener that the third stimulus matched stimulus 1. The feedback text will remain on the screen for the intertrial interval—1,000 ms in this case. This field can simply be left blank if feedback is not desired. The control files for this experiment are in the folder *wrabxwithfeedback*.

The example below shows a few lines from a stimulus-presentation file for an experiment using stereo presentation—dichotic listening in this example. The pair of signals is enclosed in braces; the first sound file will

be presented to the left channel, and the second sound file will be presented to the right channel:

```
"b d" {ba.wav da.wav}
"b g" {ba.wav ga.wav}
"b p" {ba.wav pa.wav}
"b t" {ba.wav ta.wav}
```

Examples of experiments using stereo presentation can be found in the folders *dichotic* and *duplex*. If the experimenter wants a signal to be presented to one channel only on a particular trial, a specification such as "{ba.wav blank}" can be used. See *strooplr.stm* in the *stroop* folder for an example.

The Tcl Script

The *Tcl* script essentially defines all of the features of the experiment except those that are set in the stimulus-presentation file. The *Tcl* script contains a minimum of two functions: *defineExperiment* and *layout*. Other experimenter-defined functions can be added if desired. An annotated example of the *Tcl* script *wrid.tcl*, which produced the labeling experiment shown in Figure 1, is reproduced in Appendix A.

defineExperiment

A brief summary of the most commonly used features of the *defineExperiment* function is given below:

- *-helpFile "instructions.txt"*: The text file named here will be displayed when the *Instructions* button is clicked. The file needs to be located in the experiment directory (e.g., c:\Alvin\MyExperiment) and must be a plain text file of the kind produced by simple text editors such as *Notepad* (i.e., a Microsoft *Word* document will not work).

- *-controlFile "wrid.stm"*: This gives the name of the stimulus-presentation file.
- *-shuffle 2*: This instructs Alvin to present the trials that are specified in *wrid.stm* twice in random order. Any positive integer can be used here. If the shuffle count is set to zero (*-shuffle 0*), Alvin will present the trials listed in *wrid.stm* in exactly the order in which they appear in the file. The randomization scheme that is implemented with the shuffle count scrambles stimuli *within* a block of trials; that is, once a trial is presented, it will not be presented again until all other trials in the list have been presented at least once. If randomization *across* blocks is desired, with each trial presented five times in random order, for example, simply repeat all lines in the list five times and use "*-shuffle 1*". Alvin will present all 45 trials (9 trial types × 5 presentations) in a single random order.
- *-stimulusDelay 500.0*: This specifies an intertrial interval of 500 ms.
- *-stimulusDir "signals"*: The sound files will be in a subdirectory called *signals*. For example, assume that the stimulus-presentation and *Tcl* files for an experiment are in a folder called *wrid*. The line shown here would instruct Alvin to look for the sound files in a folder called *signals* directly beneath *wrid*.
- *-replayLimit 2*: This specifies that the *Replay* button should be disabled after it is used twice. By default, there is no limit.
- *-startDelay 1000.0*: Delay in milliseconds between pressing the *Begin* button and the start of the experiment. The default is 500 ms.
- *-afterDone wriddone*: The argument that follows *-afterDone* is the name of an optional experimenter-defined *Tcl* function that is to be executed after all trials specified in the stimulus-presentation file are completed. For the line shown here, Alvin will execute the *Tcl* function *wriddone* after the experiment completes. In the present case, the *wriddone* function analyzes the data, displays the labeling function, and displays a text box with the raw data for the labeling function (see Appendix A for details). Functions such as *wriddone* can be especially handy for student laboratory assignments. The text and graphics displays that are produced here can be printed, so the student can leave the laboratory with a hard copy of his or her own data for use in preparing a report. Simply omit the *afterDone* line if there is no experimenter-defined function.
- *-beforeStart findRMS*: The argument that follows *-beforeStart* is the name of an optional experimenter-defined *Tcl* function (*findRMS* in this example) that Alvin will run prior to the start of the experiment. For examples, see *signalsinnoise.alvx* and *signalsinnoise2.alvx* in the *signalsinnoise* folder. Omit the *beforeStart* line if there is no experimenter-defined function.
- *-afterTrial resetSlider*: The argument that follows *-afterTrial* is the name of an optional experimenter-defined *Tcl* function (*resetSlider* in this example) that Alvin will run following each trial. For an example, see *grbs.tcl* in the *grbs* folder. Omit the *-afterTrial* line if there is no experimenter-defined function.
- *-formatResultLine formatLine*: The argument that follows *-formatResultLine* is the name of an optional *Tcl* function that controls the format of the Alvin output file. An illustration can be found in the folder *textexample*.
- *-presentation displayTextStimuli*: The argument that follows *-presentation* is the name of the *Tcl* function that controls the presentation of the stimuli. The example listed here is the name of the function that is used for experiments involving the presentation of text strings as stimuli. Other examples of the use of the *-presentation* option can be found in *signalsinnoise*, *mentalrotation*, *videoexample*, and *imageexample*. If this option is not used, the default function *playAudioFile* will be used. This is the audio-presentation function that is used in experiments described earlier such as *wrid*, *vowelid*, and *grbs*. In general, experimenters do not need to understand anything about how these functions work. However, if a stimulus-presentation format is needed other than one of the formats distributed with the package, it will be necessary to develop an appropriate *Tcl* function comparable to *displayTextStimuli*, *playAudioFile*, and so forth.

layout

The *layout* function controls the screen layout and defines how buttons, sliders, and text boxes will behave. Annotated examples of *layout* functions for experiments involving these three response modalities are provided in Appendixes A–C. The most heavily documented example is Appendix A, which shows the *Tcl* script for the experiment illustrated in Figure 1. Appendix B describes only the new commands that are used for sliders, and Appendix C describes only the new commands that are used for text-entry boxes. Another good source of documentation for the *layout* function is the relatively large collection of sample experiments that is distributed with the program.

Other Alvin Control Files

menu0.tcl

This plain-text file controls the experiment menu that is seen when Alvin is started (see Figure 8). Experiments can be added to or deleted from this menu by editing *menu0.tcl*. A sample entry is shown below:

```
"Vowel ID"      "vowelid/vowelid.tcl"
```

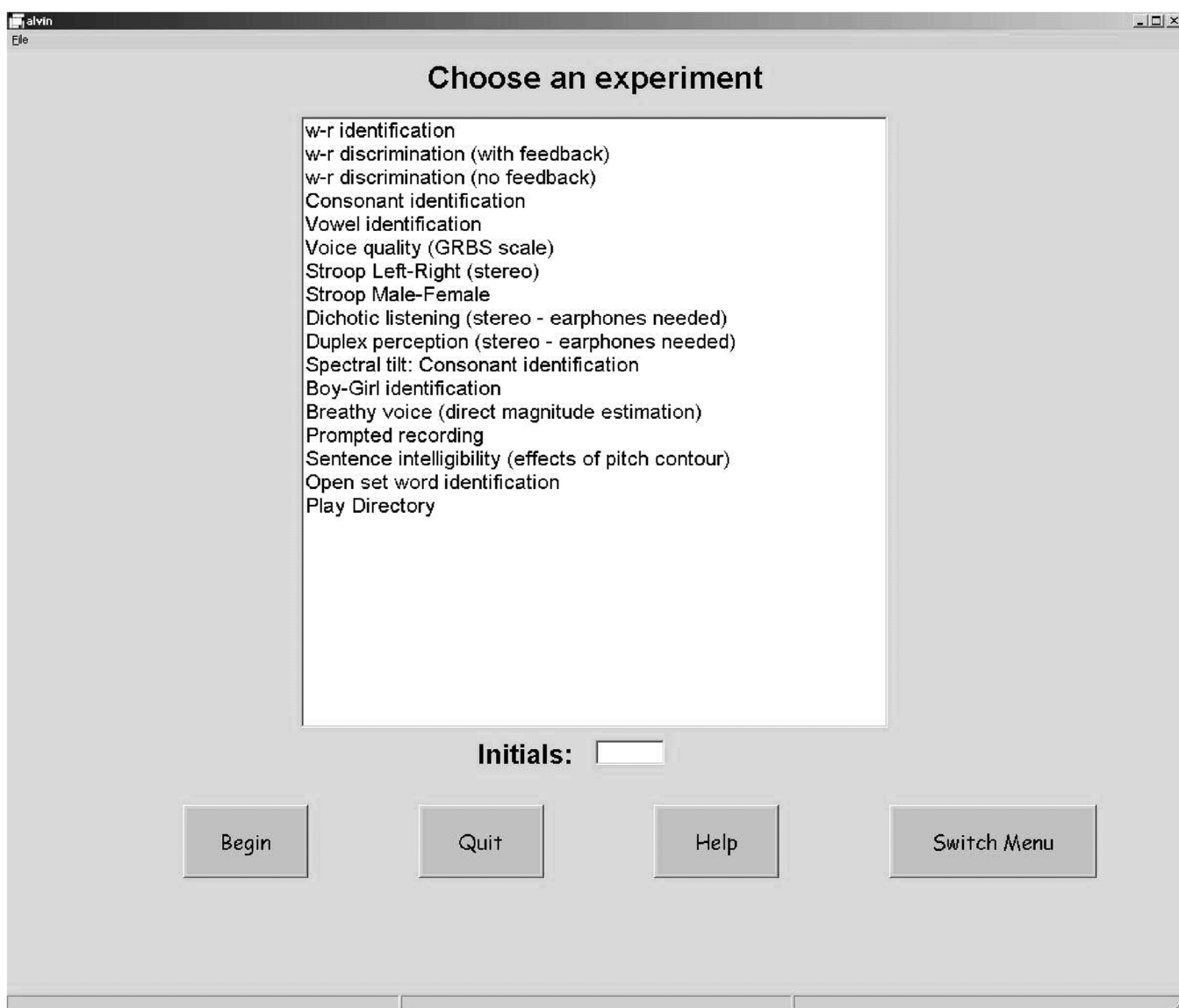
Each menu entry is defined by a single line consisting of two strings *enclosed in double quotes*. The first string is the text that will appear in the menu box, and the second specifies the location (relative to the Alvin directory) and name of the *Tcl* script that runs the experiment. For the entry above, the directory *vowelid*

needs to be located directly beneath Alvin's top directory (e.g., c:\Alvin\vowelid). The *menu0.tcl* file is located in the *tcl* folder, directly beneath Alvin's top directory.

menus.tcl

Several different menus can be maintained with Alvin (e.g., one for each experimenter). A *Switch Menu* button on the main menu allows switching among different menu files. The various menus are listed in *menus.tcl* in the *tcl* directory. A sample *menus.tcl* file is distributed with the program. Lines can be added to or deleted from this file.

Figure 8. Alvin's main menu. Participants choose an experiment from the menu, enter a participant identifier, then click the *Begin* button.



fonts.tcl

Font names that are used in the *layout* function need to be defined in the text file *fonts.tcl*. A few lines from a sample *fonts.tcl* file are shown below:

```
font .Comic -name "Comic Sans MS" -size 16
font .Courier12 -name "Courier" -size 12
font .Courier8 -name "Courier"-size 8
font .Phonetics -name "Phonetics" -size 24 -style italic
font .Symbol -name "Symbol" -size 16 -weight bold
font .Arial24 -size 24 -name "Arial" -weight bold
font .Arial20 -size 20 -name "Arial" -weight bold
font .Arial16 -size 16 -name "Arial"
```

The first text string (e.g., *.Phonetics*, *.Symbol*, *.Arial24*) associates the font specification (name, size, style, weight, etc.) with a name that can later be used in the *layout* function. Any name can be used here. The *-name* option associates this character string with a font name (e.g., *Times New Roman*, *Arial*, *Courier New*). A list of font names can be found with the Windows *Character Map* utility. As illustrated in the example, the options *-size*, *-weight*, and *-style* are used to specify other attributes of the font name. The line below shows a reference to the *Arial24* name that was defined above:

```
label .trialText -value "Trial# 0/0" -y 40 -w 100 -align center -font .Arial24
```

Characters that do not have keyboard equivalents need to be handled in a special way. The line below from a *layout* function labels a button with the 5A character from a phonetic-symbol font:

```
button .zh -value "zh" -label \\5A -command buttonClicked -font .Phonetics
```

The 5A in the *-label* option, when preceded by a double backslash, will be interpreted as the hexadecimal code for a character in whatever font is in use. The line below shows how a sequence of characters can be displayed (tj in this case):

```
button .ch -value "ch" -label \\7453 -command buttonClicked -font .Phonetics
```

Hexadecimal codes can be viewed with a variety of utilities, including Windows *Character Map* (the codes are displayed in the lower left corner). Examples making heavy use of special fonts can be found in the *vowelid* and *consonantid* folders. (Note that these two experiments, along with a few others that are distributed with the package, *will not run properly unless the phonetic symbol font is installed* using the Windows font installer [Start-Control Panel-Fonts-File-Install New Font]. The

font file itself is located in a folder called *docs* beneath Alvin's top directory [e.g., c:\Alvin\docs].)

Creating an Experiment

The list below summarizes the steps that are involved in creating an experiment with Alvin.

1. Create a folder for your experiment beneath Alvin's top directory (e.g., c:\Alvin\MyExperiment).
2. Create a *Signals* subdirectory (e.g., c:\Alvin\MyExperiment\Signals) and put your stimulus files in it.
3. Create a stimulus-presentation file in the experiment directory (e.g., *MyExperiment.stm*).
4. Create a *Tcl* script (e.g., *MyExperiment.tcl* or *MyExperiment.alvx*; see step 6 below). The easiest way to get started is to find an example that is similar to the experiment you want to run. If any fonts are used that are not already defined, add the new definitions to *fonts.tcl*.
5. If an *Instructions* button is used, create a plain text file with the instructions and put it in the experiment directory. The name needs to correspond to the one that is specified in the *defineExperiment* function of the *Tcl* script (see Appendix A).
6. Add the new experiment to *menu0.tcl*. (This file is read when Alvin starts, so it is necessary to exit and restart before any new entry is recognized.) There are two ways to launch an experiment other than (or in addition to) adding a new entry to the menu file: (a) The experiment can be run by choosing *Open Experiment* from the *File* menu (note that *Open Experiment* is different from *Open File*, which serves a different purpose, not discussed here), or (b) a shortcut to the *Tcl* file can be created and placed on the desktop or anywhere else. The experiment can then be run by double-clicking the icon. *If this method is used, the script file to which the shortcut points must end in the extension .alvx* (Alvin experiment). Regardless of how the experiment is started, the participant next enters his or her initials (or any other participant identifier), then presses the *Begin* button. With the participant identifier *jmh*, the output file from the experiment will be stored in the experiment directory as *jmh.res*. There may be additional output files, depending on how the *Tcl* script is set up.

Alvin Output Files

Alvin creates plain-text output files with a simple structure. Shown below are a few lines from an output file created by the /w/-/r/ labeling experiment shown in Figure 1:

1	1	wr01.wav	w	797
2	4	wr04.wav	r	863
3	2	wr02.wav	w	578
4	8	wr08.wav	r	625

Each line consists of (a) the trial number, (b) whatever character string appears in the stimulus-description field for the stimulus or stimuli that were presented, (c) the name or names of the stimulus files, (d) the participant's response (for a button, this will consist of whatever string was specified with the *-value* option for the button—see Appendix A), (e) the response latency in milliseconds, measured from the onset of the stimulus (or from the onset of the first stimulus if there is more than one stimulus per trial), and (f) if a *Replay* button was used, the number of times the *Replay* button was clicked. The format of the output file can be changed by specifying a *Tcl* function with *-formatResultLine* in *defineExperiment*. An example of the use of this kind of formatting function can be seen in the *textexample* experiment.

Special Feature for Presenting Signals in Noise

Since experiments involving the presentation of signals in noise are quite common, some special features have been added to Alvin to simplify their design. Shown below are a few lines from a stimulus-presentation file for a speech intelligibility experiment involving the presentation of sentences in noise:

```
{sentence1.wav noise.wav 12}
{sentence2.wav noise.wav -6}
{sentence3.wav noise.wav 3}
{sentence4.wav noise.wav inf}
```

The first entry is the name of the signal file, the second entry is the name of the noise file, and the third entry is a value in decibels that controls the signal-to-noise ratio. (Note that the noise file in this example is the same for all trials, but it does not need to be.) If the signal and noise files are equivalent in level (however defined) prior to scaling, this third value simply specifies the signal-to-noise ratio that will prevail when the signal and noise files are mixed.⁵ It is important to note that Alvin will not know whether the levels of the two files are, in fact, equivalent prior to scaling. In all cases, Alvin will scale the noise by $10^{-s/20}$, where *s* is the third entry on the line (12, -6, 3, etc., in the example above). The designation “inf” (infinite) is used when no noise is to be mixed with the signal. After scaling, the two files

⁵In our experiments, the signal and noise files are scaled to the same *rms* value, but other approaches might be taken to establishing level equivalencies. A stand-alone console application called *modrms*, distributed with Alvin, can be used to scale a list of files to a common *rms* value. A help message can be obtained by running *modrms.exe* with no arguments (e.g., *modrms<enter>*).

are simply added point-for-point. The signal and noise files need not be equal in duration, but the noise file should be at least as long as the signal file with which it will be mixed. *The signal and noise files must be sampled at the same sample frequency.* After summing the two files, the mixed signals are all scaled to a common *rms* amplitude. The *rms* value to which all mixed files are scaled is determined by Alvin when the experiment is loaded. Prior to starting the experiment, Alvin (a) reads the stimulus-presentation file, (b) mixes all signal and noise files with the specified scaling constants, and (c) determines, across all mixed files, the largest *rms* value to which all mixed files can be scaled without producing any clipping (i.e., without exceeding the limits of the 16-bit integers that are used to code instantaneous amplitude).⁶ A sample experiment that has been set up using this method is in the folder *signalsinnoise*.

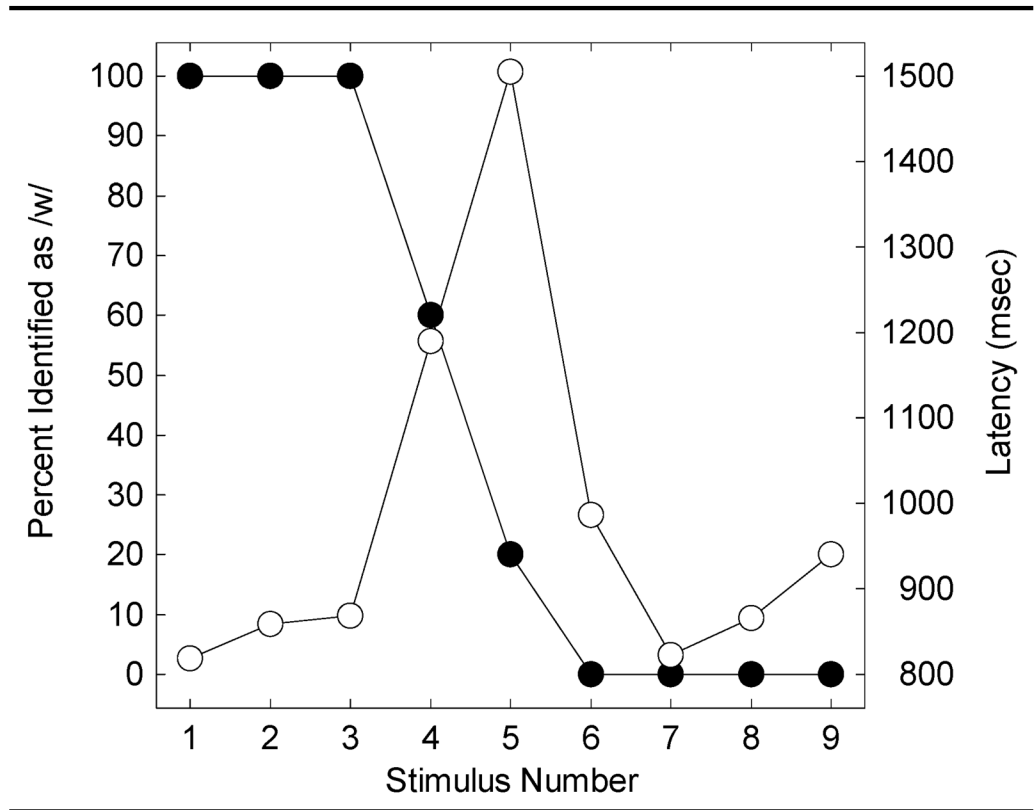
Limitations and Caveats

Adaptive procedures. Alvin is flexible and relatively easy to use, particularly if the experimental procedure is similar in format to one of the examples that is distributed with the software. While the program handles a fairly wide range of experimental procedures, the current implementation accommodates nonadaptive methods only. There is, however, nothing about the structure of Alvin that would prevent one from expanding it to accommodate adaptive techniques. The software is a combination of C++ code and *Tcl* scripts, the most important of which is a script called *basic_exp.tcl*. Since the aspects of the program having to do with experiment logic are implemented in *Tcl*, extending Alvin to accommodate adaptive techniques would involve new *Tcl* code rather than the typically much more complex C++ code.

Response latencies. While Alvin measures response latencies, Windows is not a real-time operating system, so the response latencies may not be appropriate for experimental applications in which very high precision is needed. We have not made a formal study of the accuracy of latencies reported by Alvin. We have, however, run several experiments as laboratory exercises that depend on latency data (see Figure 9). The experiments have all been replications of published studies, and we have always obtained reasonable looking results. The bottom line, though, is that the latencies that are reported may not be sufficiently precise for all applications.

⁶There is a fair amount of computation involved, so there can be a significant delay for experiments involving large numbers of sound files, particularly on slower computers. If the delay becomes problematic, the common *rms* value can be computed outside of Alvin and hard-coded into the script file. For an example and explanation of the procedure, see the comments and commands in *signalsinnoise.alux* and *signalsinnoise2.alux* in the *signalsinnoise* folder.

Figure 9. Percent identification as /w/ (filled symbols) and response latency (unfilled symbols) as a function of stimulus number for a synthetic /w/-/r/ continuum. Note the sharp increase in response latency near the category boundary. These data are from a single listener and a roughly 5-min, 90-trial listening session.



Debugging. *Tcl* was the best scripting language available at the time we began developing Alvin, but it is not without its limitations. The most important of these, in our view, is that *Tcl* offers only the most rudimentary tools for debugging, sometimes making it unnecessarily difficult to find and fix errors. *Tcl* can also be annoyingly fussy about trivia such as spacing and alignment. Plans are under way to develop a parallel version of Alvin that uses a more programmer-friendly scripting language called *Lua*.

Acknowledgments

This work was supported by Grant R01-DC01661 from the National Institutes of Health to Western Michigan University. We would like to thank Michael Clark for helpful comments on an earlier draft of this article.

Reference

MacWhinney, B., St. James, J., Shunn, C., Li, P., & Schneider, W. (2001). STEP—A system for teaching experimental psychology using E-Prime. *Behavior Research Methods, Instrumentation, and Computers*, 33, 287–296.

Received February 12, 2004

Accepted May 26, 2004

DOI: 10.1044/1092-4388(2005/005)

Contact author: James M. Hillenbrand, Department of Speech Pathology and Audiology, Western Michigan University, Kalamazoo, MI 49008. E-mail: james.hillenbrand@wmich.edu

Appendix A (p. 1 of 3). Annotated Tcl file for the two-choice labeling experiment shown in Figure 1.

```
defineExperiment {  
    -helpFile "instructions.txt"           (Display this text file when the Instructions button is clicked)  
    -controlFile "wrid.stm"              (Name of stimulus-presentation file; see above)  
    -shuffle 2                           (Present two separate scramblings of the trials in wrid.stm)  
    -stimulusDelay 500                   (Intertrial interval = 500 ms)  
    -stimulusDir "signals"               (The sound files will be in a subdirectory called signals)  
    -afterDone wriddone                   (Run the procedure wriddone after the experiment completes; optional)  
}
```

General comments. The *layout* function below controls the screen layout and the behavior of buttons, sliders, text-entry boxes, images, and text strings. Controls are available for features such as button position (-x -y), button width and height (-w -h), text labels on the buttons (-label and -label2), optional keyboard shortcuts (-key), and the name of the procedure to be run when the button is clicked (-command). By default, screen positions are specified as percentages of screen width and height. Absolute screen coordinates can be specified by following the numeric value with the characters *px*; for example, -x 20 will be interpreted as 20% in from the left edge of the screen, while -x 20px will be interpreted as 20 pixels from the left edge. The *-value* option specifies a text string to be written to the output file when the button is clicked. In the example below, the character *w* will be written to the output file when the *.w* button is clicked, but this can be more elaborate if desired. This text field is used to specify the attributes of the response in the same way that the stimulus-description field in the stimulus-presentation specifies the attributes of the stimulus. For example, if desired, codes might be added to this text field to specify the place, manner, and voicing features of the response, which could then be compared at analysis time with the corresponding features of the stimulus. Two final notes: (a) *Tcl* is case sensitive and sometimes aggravatingly fussy about spaces and the alignment of braces, and (b) the intent in allowing the specification of screen position as percentages was to make the general look of the display reasonably independent of display settings. For various reasons, this does not always work. The examples that are distributed with the software were created using a fairly large monitor set at 1600 × 1200. The displays may look pinched, and text may overlap with buttons when the experiments are run on monitors set to lower resolution. This can be fixed with minor changes to the *layout* function:

```
proc layout {} {
```

The command below sets the background color. You can get a guide to the color names by starting Alvin and, under the File menu: (a) Open Experiment, (b) open the *colorscheme* folder under Alvin's top directory, and (c) choose *colorscheme.tcl*. These same color names can also be used with buttons to set the background and text colors with the *-backcolor* and *-textcolor* options, and text colors with the *label* command using the *-color* option:

```
    canvas backcolor "forest green"
```

The *default button* commands below specify default settings for the buttons. These can be used to avoid repeating the height, width, font, and so forth for a set of very similar buttons. The defaults can be updated at any time. The commands below set the default height and width to 7 (-w and -h), where 7 means 7% of screen height (or width). The *-font2* option sets the default font for *label2* (see below) to Comic (see the description of *fonts.tcl* above); *-command buttonClicked* means that, by default, the function *buttonClicked* will be called when the button is clicked. The *-backcolor* option sets the background color of the button, while *-textcolor* sets the color of the text label:

```
    default button -w 7 -h 7 -command buttonClicked -font2 .Comic
```

```
    default button -backcolor "lightblue" -textcolor "black"
```

The line below places a button called *.w* 42% in from the left edge of the screen and 70% up from the bottom edge with a nondefault height of 9%. Look at the text labels on the two buttons shown in Figure 1 to see what the *label* and *-label2* options do. The text that appears in the *-value* field will appear in the output file whenever the participant presses this button. This can be a character string rather than a single character. This field is used to specify the attributes of the response in the same way that the stimulus-description field specifies the attributes of the stimulus. See *consonantid.tcl* for a more elaborate use of the *-value* option. The *-key* option allows a key to be associated with the button; that is, pressing the *w* key on the keyboard will have the same effect as clicking the button:

```
    button .w -x 42 -y 70 -h 9 -label W -label2 way -key w -value "w"
```

```
    button .r -x 50 -y 70 -h 9 -label R -label2 ray -key r -value "r"
```

Appendix A (p. 2 of 3). Annotated Tcl file for the two-choice labeling experiment shown in Figure 1.

The line below places text on the screen, aligned in the center and 40% up from the bottom. The text string will report the trial number as the experiment proceeds:

```
label .trialText -value "Trial# 0/0" -y 40 -w 100 -align center -font .Arial24
```

This line resets the button defaults:

```
default button -w 14 -h 7 -y 30
```

The definitions of the *Start*, *Back Up*, *Instructions*, and *Return to Main Menu* buttons below are fairly standard, with the possible exception of positioning:

```
button .start          -x 8  -label "Start"          -command startClicked -key s
button .backup         -x 26 -label "Back Up (b)"      -command backupClicked -key b
button .help           -x 44 -label "Instructions"    -command helpClicked
button .viewresults    -x 62 -label "View Results"    -command viewresultsClicked
button .return         -x 80 -label "Main Menu"     -command returnClicked
```

```
enable .replay 0      (Disable the Replay button; it is enabled automatically when the experiment begins)
```

```
enable .backup 0      (Ditto for the Back Up button)
```

```
enable .return 1      (Enable the Return button)
```

```
enable .viewresults 0 (Disable the View Results button; this will be enabled below in wriddone)
```

```
}
```

Run the function below when the button labeled *View Results* is clicked (this is optional):

```
proc viewresultsClicked { buttonName } {
```

Make the output filename for the analysis program *res2id.exe*. If the results file is *jmh.res*, the line below will create an output file called *jmh.id*; [resultFile] below is a function that returns the name of the *.res* file:

```
set outfile [changeExtension [resultFile] ".id"]
```

Execute *res2id.exe* (in the experiment directory), which will read the *.res* file and create a labeling function from it:

```
exec res2id [resultFile] c01=w c02=r nstim=9 quiet scol=2 rcol=4 ncol=3
```

Take a look at the labeling function as a text file:

```
viewfile $outfile -x 5 -y 95 -w 85 -h 85
```

```
}
```

Run the function below when the experiment finishes (this is optional):

```
proc wriddone {} {
```

```
enable .viewresults 1
```

If the results file is *jmh.res*, the line below creates the string *\$outfile* as *jmh.id*, which will be displayed with the *viewfile* command below:

```
set outfile [changeExtension [resultFile] ".id"]
```

Execute *res2id.exe*, which will read the *.res* file and create a labeling function from it:

```
exec res2id [resultFile] c01=w c02=r nstim=9 quiet scol=2 rcol=4 ncol=3
```

Take a look at the labeling function:

```
viewfile $outfile -x 5 -y 95 -w 85 -h 85
```

Plot the labeling function using the procedure *plotidfunction*:

```
frame .plot -x 25 -y 75 -w 70 -h 70 -draw plotidfunction -title Plot
```

```
}
```

```
proc plotidfunction { } {
```

Plot the labeling function:

```
set outfile [changeExtension [resultFile] ".id"]
```

Appendix A (p. 3 of 3). Annotated Tcl file for the two-choice labeling experiment shown in Figure 1.

Open the .id file with the labeling function:

```
openfile $outfile
```

Set the dimensions of the graph: *b*, *t*, *l*, and *r* stand for bottom, top, left, and right. The values are in percentages of the screen width and height; that is, *l*=15 means set the left edge 15% in. (Use "axes=4" to see what this option does.) Note that *jset* is used here rather than *set* because *set* is a reserved word in Tcl:

```
jset axes=2 b=15 t=85 l=15 r=75
```

Specify text labels for the axes:

```
xlabel Stimulus Number
```

```
ylabel Percent Identified as /w/
```

The line below specifies an *x*-axis with a maximum value of 9 and a minimum value of 1, with tick marks and numeric labels every 1 unit:

```
xlimit 9 1 1
```

The line below specifies a *y*-axis with a maximum value of 100 and a minimum value of 0, with tick marks and numeric labels every 10 units:

```
ylimit 100 0 10
```

Linegraph command below: Plot column 1 (*x*=1) against column 3 (*y*=3); *symbol=C* specifies a filled circle. There are many choices here (a few others: *T* = filled triangle, *S* = filled square, *D* = filled diamond). Lowercase letters (e.g., *symbol=c*) are used for unfilled symbols. See *symbol.cpp* in the *src/jgraf* source code directory for a complete set; *size=8* makes an 8-pixel symbol; *color=red* makes a red line with red symbols. The symbol and line colors can be set separately with a command like "*lcolor=red scolor=blue*":

```
linegraph x=1 y=3 symbol=C size=8 color=red
```

Note that there are Print, Copy to Clipboard, and Save BMP File options under the File menu that can be used once the graph is plotted.

```
}
```

Appendix B. Annotated Tcl file illustrating the use of a slider.

```
defineExperiment {  
-helpFile "instructions.txt"  
-shuffle 1  
-controlFile "sliderexample.stm"  
-stimulusDir "signals"  
}
```

```
proc layout { } {  
canvas backcolor "pink"  
default label -h 5 -font .Arial16  
Label the slider endpoints with the words "Smooth" and "Rough":  
label .sliderLeft -value "Smooth" -x 25 -y 60 -w 10  
label .sliderRight -value "Rough" -x 65 -y 60 -w 10
```

Position a slider 35% in from the left, 70% up from the bottom with a width of 40%. Assign a value of 0 to the left edge of the slider and a value of 1000 to the right edge. Give the slider arrow an initial position of 500 at the start of the trial. Note that the arrow is moved by clicking a spot along the length of the slider, not by dragging the arrow. See the *grbs* folder for an example illustrating the use of several sliders to rate different aspects of the stimulus:

```
slider .resultSlider -x 30 -y 55 -w 40 -h 5 -initial 500 -min 0 -max 1000
```

Put up a button labeled *Okay* that the participant uses to terminate the trial (the *Enter* button on the keyboard serves the same purpose):

```
button .okay -label "Okay" -x 80 -y 57 -w 14 -h 0 -command sliderEntered  
label .trialText -value "Trial# 0/0" -y 40 -w 100 -align center  
default button -w 14 -h 7 -y 30
```

```
button .start -x 10 -label "Start" -command startClicked -key s  
button .replay -x 30 -label "Replay (r)" -command replayClicked -key r  
button .backup -x 50 -label "Back Up (b)" -command backupClicked -key b  
button .return -x 70 -label "Return to Main Menu" -command returnClicked -w 23
```

```
enable .resultText 0  
enable .replay 0  
enable .backup 0  
enable .return 1  
}
```

```
proc sliderEntered { buttonName } {  
# use a catch to ignore clicks before we've started  
catch {  
set value [.resultSlider value]  
setvalue .resultSlider 500  
storeResult $value  
}  
}
```

Note. Commands that are not documented here are explained in Appendix A.

Appendix C. Annotated Tcl file for illustrating the use of a text-entry box.

```
defineExperiment {
  -name "br"
  -helpFile "instructions.txt"
  -shuffle 2
  -controlFile "br.stm"
  -stimulusDir "signals"
  -afterDone sendDoneMessage
}
```

```
proc layout { } {
```

```
  canvas backcolor "green yellow"
```

```
  label .label0 -value "Enter a number as large as the signal is breathy." -x 5 -y 65
```

```
  label .resultLabel -value "Enter breathiness estimate:" -x 30 -y 55 -font .Arial16
```

The *text* command below positions a text entry box 55% in from the left, 55% up from the bottom, with a width of 10% and a height of 20%. The *-allow* option tells Alvin to ignore any nonnumeric text:

```
  text .resultText -x 55 -y 55 -w 10 -h 20 -command textEntered -allow "01234567890"
```

Put up a button labeled *Okay* that the participant uses to terminate the trial. As with sliders, the *Enter* button on the keyboard can be used in place of clicking the *Okay* button:

```
  button .okay -label "Okay" -x 70 -y 57 -w 14 -h 0 -command okayClicked
```

```
  label .trialText -value "Trial# 0/0" -y 42 -w 100 -align center -font .Arial16
```

```
  default button -w 14 -h 7 -y 30
```

```
  button .start -x 10 -label "Start" -command startClicked -key s
```

```
  button .replay -x 30 -label "Replay (r)" -command replayClicked -key r
```

```
  button .backup -x 50 -label "Back Up (b)" -command backupClicked -key b
```

```
  button .return -x 70 -label "Return to Main Menu" -command returnClicked -w 23
```

```
  enable .resultText 0
```

```
  enable .replay 0
```

```
  enable .backup 0
```

```
  enable .return 1
```

```
}
```

```
proc okayClicked { buttonName } {
```

```
  # use a catch to ignore clicks before we've started
```

```
  catch {
```

```
    textEntered .resultText
```

```
  }
```

```
}
```

Note. Commands that are not documented here are explained in Appendix A.
