Automatic Volume Control for Auditory Interfaces

Stephen Brewster, Andrew Crossan and Murray Crease

Glasgow Interactive Systems Group,
Department of Computing Science,
University of Glasgow, Glasgow, G12 8RZ, UK.

Tel: +44 (0)141 330 4966
Fax: +44 (0) 141 330 4913
Email: [stephen, ac, murray]@dcs.gla.ac.uk
Web: www.dcs.gla.ac.uk/~stephen

1 Introduction

The use of sound output (speech and non-speech sounds) to present information is an active area of research within multimodal HCI. Sound is used in a range of devices from desktop computers, personal digital assistants and mobile telephones, to domestic appliances (such as microwaves that use sound to indicate when cooking is complete). One major problem with all such devices is that the sounds are often played at the wrong volume. The manufacturers do not know the ambient noise levels of the environments where devices will be used and so take a 'better safe than sorry' approach and make sounds loud (e.g. loud mobile phone ringing sounds in quiet cinemas). Excessive loudness of sounds is a major cause of annoyance (Berglund *et al.*, 1990). Alternatively, in very loud environments sounds may be played too quietly and masked - rendering the information they contain unusable. The simple solution described here is to automatically control the volume of sounds to ensure they are played at just above the ambient sound level – they will always be at a suitable volume.

Automatic gain control is common in the music/audio recording industry where recording level is altered in response to the amplitude of the sound being recorded. If the input signal is quiet then the gain is increased so that it will be heard on the recording and *vice versa*. Many computing devices (desktop PCs, mobile telephones and PDAs) now have built-in microphones and loudspeakers so that, here too, we have the possibility of monitoring the ambient sound volume and adjusting the sound level out of the speakers appropriately. What is needed is the control software.

2 The automatic volume control system

Our system is built in Java and uses the JavaSound package. It currently runs on a desktop PC with a standard SoundBlaster soundcard and microphone (it can run on any platform that supports Java 1.3 and that has a microphone and speakers). It runs in the background, monitoring the ambient sound level in the environment in real time, and changes the level of the sounds output from the loudspeakers accordingly.

There are two main components. The *Sampler* uses the microphone to sample the ambient sound level for 0.5 secs. in every 1.5 secs. The amplitude of the signal is then extracted from each of the samples. The samples are 8kHz, 8 bit linear encoded sounds. High sampling frequency/bit rates are not needed as only an average volume over the period of the sample is required (this also reduces the amount of computation that needs to be done). The sampler generates a weighted average of the volume in each of the samples to pass on to the next part of the system. It takes an average of the current sampled value and the 5 previously sampled values to generate the new value. This allows us to smooth out any short, sharp peaks in the sound level.

The *Audio Level Controller (ALC)* maintains the value of the current output volume. It adjusts this smoothly over a period of time based on the difference between the weighted average from the *Sampler* and the current output volume level. The smoothing function used adjusts the rate at which the output level changes in response to an increase in the average from the *Sampler*; Adjusting faster for larger differences, for example, reacts more quickly to large changes in ambient volume. This smoothing function again reduces drastic fluctuations in the volume of sounds being played out of the speakers to avoid annoying users. A conservative approach was taken so that the volume changed only when there was an increase in the ambient sound level over a longer period. The disadvantage of this was that it could result in some sounds being played too quietly as the sound output level had not yet been increased. The advantage was that sharp fluctuations (and so annoyance) were avoided.

The user can also adjust the level at which sounds will be played above the ambient threshold (done in our system by using a volume slider). This is then used as an offset in the *ALC's* calculations. For example, one user might have a hearing difficulty that means he/she needs sounds played louder than another does.

One problem that must be avoided is the feedback loop. If the device raises its volume the sound in the environment around it may become louder, therefore the next time the system samples it would want to raise its volume again and so on, making a feedback loop which would result in sounds being played at maximum volume. The sounds used in our own auditory interfaces are discrete, with most lasting less than two seconds (Brewster, 1998)) and are therefore generally not a problem due to the smoothing that is done. To further avoid the problems (especially with continuous sounds) the microphone was positioned to point in the opposite direction to the speakers. The standard SoundBlaster microphone is highly directional – if it is not pointing directly at the speakers it is almost impossible to put our system into a feedback loop. In addition, the *Sampler* also rounds down the average sample values it produces. There is again a risk that in

doing this the sounds will be played too quietly but we chose this more conservative approach to avoid the risk of annoyance highlighted by Berglund *et al*.

The system has been initially tested and shown to be effective in a range of office environments – silence, with music playing, with talking/office noise in the background. The output volume of the system changes reliably in response to changes in the ambient sound level. The measures taken to avoid feedback have been successful (in fact it is very difficult to put the system into a feedback loop even if you really try!). We have also developed this to work for adjusting screen brightness. Using a QuickCam, the visual version checks the ambient brightness of the environment around the computer and adjusts the screen brightness accordingly to avoid eyestrain.

3 Conclusions

The problems that can occur because sound output from auditory interfaces is at the wrong volume can be solved by monitoring the ambient sound level around the device making the sound and then setting the device's output volume appropriately. Reducing the problems of annoyance due to excessive volume of sound is a very important step in increasing the acceptability of auditory interfaces. With our software this can now be done on a wide range of different computing devices using their existing hardware.

Acknowledgements

This work was supported by EPSRC Grant GR/L66373. Thanks to Tahir Khan for his help in this project.

References

Berglund, B., Preis, A. & Rankin, K. (1990). Relationship between loudness and annoyance for ten community sounds. *Environment International*, **16**, 523-531.

Brewster, S.A. (1998). Using Non-Speech Sounds to Provide Navigation Cues. *ACM Transactions on Computer-Human Interaction*, **5**, 224-259.