A SYSTEM FOR REAL-TIME VIRTUAL AUDITORY SPACE

Jacob W. Scarpaci
H. Steven Colburn

Department of Biomedical Engineering and
Hearing Research Center
Boston University
Boston, Massachusetts 02215
scarpaci@bu.edu

John A. White

Department of Biomedical Engineering and
Center for BioDynamics
Boston University
Boston, Massachusetts 02215

ABSTRACT

The understanding of how people perceive spatially dynamic sound sources is limited, due in part to the difficulty of controlling dynamic environmental interactions. Free field solutions require large rooms with bulky equipment for moving speakers. Virtual systems that account for head movement must balance adaptability with stability of timing characteristics (time lag and jitter). The system described here combines the flexibility and ease of use of a software system, with near-hardware stability of a real-time operating system. Hardware and software design choices are discussed and the system performance is evaluated. Preliminary psychophysics are run to validate the system and to illustrate the need to study system latencies smaller than previously considered relevant.

1. INTRODUCTION

The study of spatially dynamic virtual sound localization is difficult to implement. The head related transfer functions (HRTFs) change as the subject moves his/her head. Therefore the subject's interaction in the environment must be considered in real time. In order to create the virtual environment, the system must be able to change filters and render the output with minimal time lag. The system described here is meant for installation in a psychoacoustics laboratory for experiments that explore thresholds of auditory perception. These experiments require the system to be flexible, accurate, and easy to use. Although the use of specialized hardware such as DSPs or FPGAs may produce more stable results, their flexibility requires specialized programming skills not often found in a psychoacoustics lab. Furthermore, streaming file I/O is more complicated to implement on specialized hardware.

The goal of the Real Time Virtual Auditory Space (RTVAS) system is to generate dynamic acoustic stimuli with a high performance, cost-effective, flexible system that can dynamically update filter coefficients in real time. These stimuli will enable investigators to explore dynamic localization cues that arise from movement of the source and/or of the subject's head. RTVAS is written entirely in C++ and runs on stock computer hardware. This allows for code to be updated quickly and real-time streaming of data to and from the disk to be implemented easily. The system is open source and uses readily available hardware, allowing other labs to implement and modify this technology with minimal effort. Previous work with virtual environments utilizing head tracking has been reported by Wenzel [1], Wightman and Kistler [2] and Brungart et al. [3]. The last of these studies showed no significant effects of system latencies smaller than 70 ms in a statically located sound localization task. Some of the work reported here has been previously reported by Scarpaci and Colburn [4].

2. SYSTEM HARDWARE

The RTVAS system is comprised of standard components, allowing for easy implementation. Currently the RTVAS system is being run on a stock Dell OptiPlex with a 2.6 GHz Intel Pentium Processor and 512 Mb of DDR RAM.[1] The operating system is a Linux 2.4.20 kernel with a Real-Time Application Interface (RTAI) real-time kernel patch.[2]

The Data Acquisition Card (DAQ) is a National Instruments NI PCI-6052E card.[3] This card has 16 analog inputs and 2 analog outputs with 16-bit resolution and up to a 333 kHz sample rate. The range of the output channels is $\pm$10V, and each sample is displayed on the output at a rate of 44.1 kHz by our real-time process. This sample rate is set in software and can be easily changed to fit experimental design. RTVAS can utilize the A/D channels on the DAQ in a variety of different ways including obtaining signal input and subject feedback. The Digital I/O can also be utilized for external triggering as well as user feedback by means of a response button. The drivers used to control the DAQ are open source Comedi drivers.[4] The Comedi drivers are supported by RTAI and control a wide range of readily available hardware. This allows our system to run in a variety of hardware configurations. Comedi's interface is straightforward and easy to learn and implement for someone competent in C/C++ programming.

Head position is measured with the InterSense IS-900*VWT* Precision Motion Tracker.[5] This system allows for 6 degrees of freedom (yaw, pitch, roll, X, Y, and Z) and can track up to 4 devices at a time. The two devices that may be used in RTVAS are the head tracker and wand. Both of these devices have gyroscopic sensors as well as ultrasonic sensors that pick up signals from beacons placed in the room. The processing needed to compute the position of each device is implemented in the InterSense hardware. The hardware then transmits position data to the computer through the serial port.

[1] Dell Inc. http://www.dell.org
[2] Real-Time Application Interface http://www.rtai.org
[3] National Instruments http://www.ni.com
[4] Linux Control and Measurement Device Interface http://www.comedi.org
[5] InterSense Inc. http://www.intersense.com

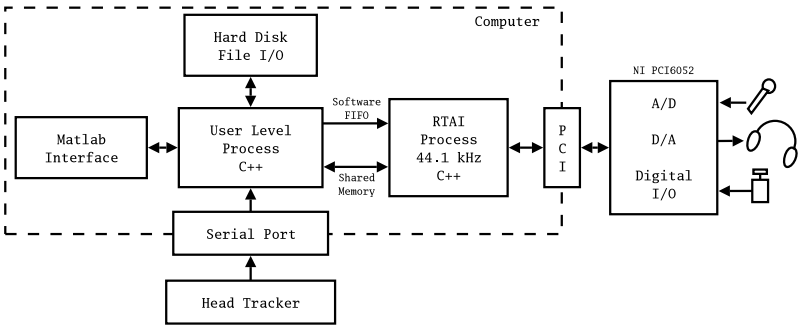

Figure 1: System level flow diagram for the RTVAS system. Subject hears output over headphones and the system receives feedback through the head tracker and response button.

The operating system, RTAI real-time patch, and DAQ drivers are all open source and freely distributable. Since all of the hardware is standard, any component of the system can be upgraded or changed without affecting the other components. This allows the system to be completely scalable; e.g. more complicated algorithms can be implemented by simply upgrading the computer.

3. SYSTEM DESIGN

3.1. Real-Time Implementation

The RTVAS system utilizes two real-time methods in order to stabilize timing. The first is a soft-real-time scheduler native to Linux called the FIFO scheduler. This scheduler preempts most other processes running on the operating system, including operating system processes that handle mouse movements and refreshing the screen. These processes would normally take processor time away from the running virtual display system, causing time jitter. Using this method the real-time process can still be interrupted by kernel level processes and is thus still considered soft-real-time. The RTAI kernel patch allows for hard-real-time performance by disabling all maskable interrupts and running the process with kernel-level priority. Furthermore this scheduler is truly periodic, interrupting any currently running process when the real-time process is scheduled to run.

Using RTAI for real-time research has been implemented by one of the authors [5]. Even though the application is real-time patch clamping instead of real-time virtual audio, the use of RTAI is proven to be a stable and effective method for obtaining hard-real-time performance.

3.2. High level system structure

Some of the advantages to a software based system are the flexibility and control over data structures. Using a computer also allows for disk and serial I/O to be straightforward. Furthermore the use of a standard operating system allows for a Matlab interface, making it accessible for users of many skill levels. The overall system is shown in Fig. 1.

The Matlab user interface sets up variables and creates input and position vectors. This part of the system is non-real-time and is not available during time-critical periods. The Matlab interface allows for easy scripting of dynamic experiments with intuitive user response and feedback utilizing graphical user interfaces.

The user level process parses the command line and sets up the auditory objects described below. During run time the soft-real-time threads running in user space control serial I/O and streaming of data to and from the hard disk. Data is shared with the hard-real-time process by means of shared memory and software FIFOs. This shared memory allows for the user space program to configure the hard-real-time process during run time.

The RTAI Process handles all of the computation of output as well as I/O to the data acquisition card. These processes are time sensitive; thus it is important to run them in a hard-real-time environment.

A multi-threaded approach allows for the most efficient use of processor time. The file I/O and serial routines are blocking processes which means there are times where these processes are waiting for data to be received by the processor. During those wait periods, it is important to give control of the processor to one of the other threads running in the system. File I/O and serial I/O are implemented in soft-real-time to allow multi-threading.

3.3. Software Architecture

The system is implemented using an object-oriented architecture. The class structure shown in Fig. 2 allows the system to be dynamic and easily expanded. To add a class or change a model, one needs only to create an inherited class and implement the functions described by the generic function headers, therefore the core of the program only sees the generic classes and their function headers. It is blind to the implementation of the virtual functions.

The object-oriented approach also allows multiple auditory objects to occur simultaneously in the auditory space. Each auditory object has separate memory space for state variables and output data, and separate member classes for obtaining input, position, and for calculating output. These classes are derived from parent classes GenericInput, GenericPosition, and GenericCalc. The generic classes are the only interface between the system and the derived classes. Each of the derived classes must be self-contained and have a defined rt_get() or rt_calc() function. To add an object to the system, pointers to the class instances are passed to the add_obj() function. For example:

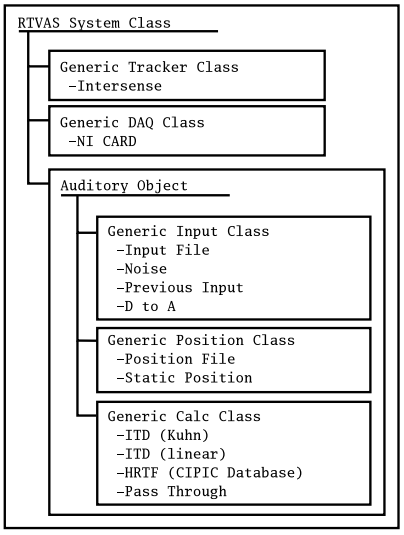

Figure 2: Class structure. Derived classes are shown connected to their parent classes. Virtual function headers defined in generic classes must be implemented in derived classes.

```
add_obj(new inputnoise(),
   new positionfile(''pos.dat''),
   new calcFilt(''Filter.dat''));
```

This allows the system to be configured at run time rather than recompiling every time there is a change.

In addition to having multiple sources, the system can be configured to add reflections by modeling them as separate sources with a delayed and filtered version of the same input. By implementing the reflections in this manner, one can study reverberant conditions without using extremely long filters. This allows these filters to be implemented in real time.

As mentioned above, this architecture allows researchers to implement their own model without having knowledge of the system architecture. A user may simply write a new class, inheriting the GenericCalc class. Within the implemented class, the virtual function rt_calc is passed all the relative data needed for the computation of the output. The complexity of the models used is solely limited by processor speed. Since the system is scalable, the computer may be upgraded and more complex models can be implemented without changing the system architecture.

4. IMPLEMENTED OUTPUT ALGORITHMS

4.1. HRTF Model

An example of an HRTF model currently implemented is that using the CIPIC HRTF database [6]. The HRTF class filters the input with 200-point anechoic head-related impulse responses (HRIRs) stored in memory. These HRIRs are taken from the 1250 positions recorded from the KEMAR mannequin that are part of the CIPIC database. However, the class is flexible and can be passed any properly formatted file of HRIRs at any temporal or spatial resolution.

The measured data sets that we are using are not spatially-sampled densely enough, which may cause "clicks" when changing between adjacent filters. We have implemented and are currently evaluating different methods for interpolation. For the discussion in this paper we will use a method similar to that described by Kistler and Wightman [7]. Although the initial work of interpolating the HRIRs has been done offline, we hope to implement the interpolation in real time.

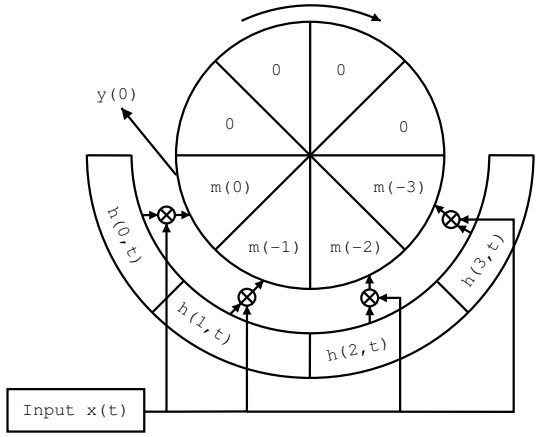

Figure 3: The current input sample is multiplied by the current impulse response and added to circular output buffer. Y(0) is added to the system output, the output buffer is shifted, the impulse response h is updated, and procedure is repeated.

4.2. Convolution Engine

Due to dynamic filters and concerns about system latency, an "output side" algorithm that accounts for time dependent filters is used. If FL is the length of the filter, the output side algorithm is as follows:

$$y(i) = \sum_{k=i-FL+1}^{i} h(i-k,k)x(k) \qquad (1)$$

In this equation $h(n,m)$ is the response n time steps after an impulse which occurred at time m. This equation allows the current output to depend on past input samples filtered by the system as it existed when the sample occurred.

Calculating one point of output per cycle of the real-time thread eliminates latencies induced by block convolution. Since the filters change with time, a circular output buffer is used to store the tails of past scaled impulse responses to be used in the current output sample (see Fig. 3). Block convolution may be used to allow for longer filters or more complicated algorithms that take more than a sample period to compute. Each sample in the block convolution will add an additional sample period (23 μs) latency to the system.

4.3. ITD Model

An important cue for localization of sound is the Interaural Time Difference (ITD). This is the difference in time it takes a signal to reach one ear relative to its arrival at the other ear. While working on the issues of implementing the HRTF models, we decided to also explore a much simpler output algorithm. By simply delaying one ear relative to the other we implemented a simple ITD model. This model ignores any frequency dependence of the ITD,

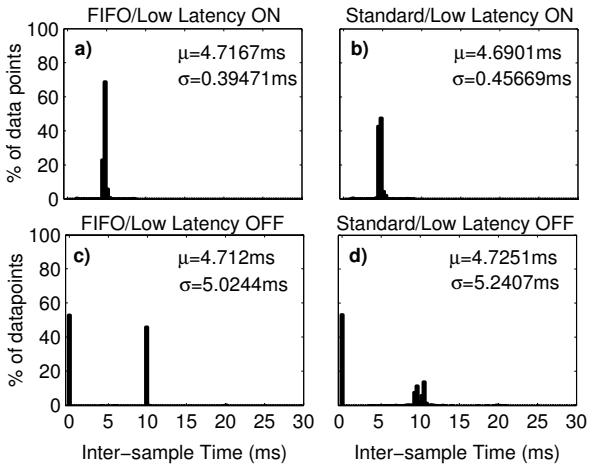

Figure 4: Timing characteristics of 10,000 packets received by the serial port. FIFO scheduling used in panels a and c allows for soft-real-time processing. Low latency serial mode used in panels a and b reduces jitter by instructing the serial port hardware to interrupt the processor for every byte received. Panels c and d show a bi-modal distribution near 0 and 10 ms due to the low latency mode being disabled.

to which subjects have been shown to be insensitive [8]. We took the azimuthal angle of the egocentric source position and used it to calculate the ITD using the low frequency ITD model described by Kuhn [9]. Half the ITD is imposed on one ear as a lead and half is imposed on the other ear as a lag.

It has been shown by Mills [10] that the minimal audible angle (i.e. the resolution at which a subject can distinguish a spatially separated signal in the azimuth) can be as small as 1 degree in the mid-line. For an average sized head this corresponds to a difference in ITD around 10 μs, which has been shown to be the just noticeable difference (JND) in ITD at the midline [11]. At a commonly used sampling freq of 44.1 kHz the smallest integer sample delay that can be implemented is 23 μs. In order to make delays small enough to account for human sensitivity of 10 μs it is necessary to use methods to delay the signals less than a sample period. To implement this delay we filter the signal with a sinc function as described in Laakso et al. [12].

5. SYSTEM PERFORMANCE

Two important metrics to consider when evaluating a real-time system are the latency and time jitter of the feedback updating the output. First we consider the latency and time jitter created by the head tracker system.

The largest source of system latency is the head tracker. Some of this latency is due to the head tracker hardware and some is due to the transmission of the data through the serial port of the computer. Figure 4 shows the timing characteristics of the serial communication. We have taken two steps to reduce the transmission-induced jitter. The first step is to run the system in a soft-real-time mode (FIFO scheduling) which is a standard Linux scheduler that has priority over normal system tasks. The timing characteristics using FIFO scheduling are shown in panels a and c. Under normal scheduling other processes can block the communication process

Movements	Trials	System Load	Mean (ms)	Standard Error (ms)
Similar to fast head movements	75	Low	6.31	0.26
	75	High	6.59	0.22
Faster than fast head movements	25	Low	10.41	0.14
	25	High	10.41	0.01

Table 1: Results of tracker latency experiment. Total system delay between head tracker movement and response of the system to the movement.

causing time jitter, shown here in panels b and d as the spread of the inter-sample time histogram. The second step is to put the serial port hardware into low latency mode, as shown in panels a and b. In this mode the serial port hardware will notify the system processor after every byte is received. In the normal mode, the hardware buffers the data and notifies the processor only after multiple bytes have been received. This will effectively skip tracker samples since the next position sample will be handled immediately after the first sample. This is shown in panel c and d as a bimodal distribution with centers at 0 and 10 ms. Under normal operation we will run the system with both FIFO scheduling and low latency mode enabled.

We have also attempted to characterize the timing characteristics of the head tracker hardware. According to the InterSense documentation the tracker should have a latency between 4 - 10 ms [13, p. 93]. To validate this, we took our own measurements comparing the tracker data to data recorded from an independent device which could measure azimuth in real time. These measurements represent the total system timing latency including the head tracker, serial communication, computer processing, and DAQ.

The two independent measures of azimuth as a function of time were compared by taking the cross correlation and finding the peak. This measurement was taken for a high computational load (file I/O for input and position as well as a 200-point convolution for each output sample), and for a low computational load (no file I/O or output calculations). The mean and standard error of the delays are shown in Table 1. Movements were made by hand by rotating the tracker's headpiece in azimuth around the post of the measurement device.

No statistical difference is seen between low and high system loads, which validates our assertion that the timing characteristics are dominated by the head tracker. For movements similar to fast head movements, the timing characteristics seem to be well within the specifications of the InterSense documentation of 4-10 ms. For movements that are faster than what a subject would be able to make, the timing characteristics were outside specifications, although only slightly.

Additional latency and time jitter due to the RTAI system is very small compared to that caused by the tracker. The hard-real-time function controlled by the RTAI scheduler is called every 22.68 μs. Fig. 5 shows the deviation from that sample period. Even though the inset shows timing overshoots as large as 15 μs, the error histogram shows that 95% of the error is within 1.56 μs. The time jitter is always less than a single sample period. This implies that with hardware buffering of one sample there would be no time jitter in the output displayed to the subject.

The implication of these findings is that the system is currently limited by the head tracker hardware. Theoretically, the system can update the filters every sample period with only one sample of

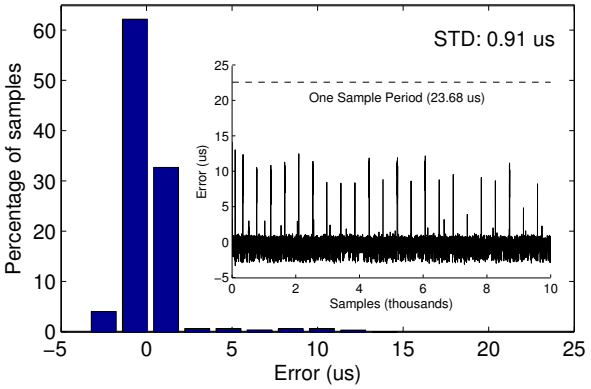

Figure 5: Histogram of the error between when a time step should have occurred and when it actually occurred. Figure inset shows a sample of the error, 95% of the error is within 1.56 μs.

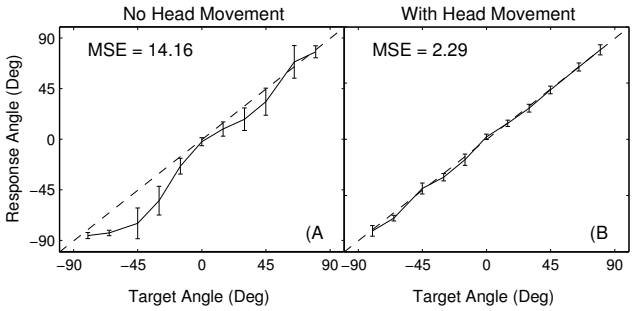

Figure 6: Results of 1 subject in a localization task. Panel A shows target vs response angle for static localization task. Panel B shows results for localization using head movement.

latency, and effectively no time jitter. The modularity of the system allows for the head tracker hardware to be easily updated to bring the timing characteristics of the system closer to its theoretical limit.

The computational complexity of the models also influences temporal stability. The number of auditory objects that the system can render while streaming input and position data from disk and recording timing information to disk was measured. Results show that while maintaining temporal stability, the system can render 4 auditory objects using the ITD model or 1 object using the larger HRTF filter model. More objects can be rendered in real time by decreasing the constraints on the system latency or by increasing processor speed.

6. PRELIMINARY PSYCHOPHYSICS

6.1. Basic Characterization of Sound Localization

To validate the effectiveness of the system for spatialization we ran localization experiments with stationary sound sources. For the static condition, subjects were played a 2-second lowpass noise stimuli with a cutoff of 1.5 kHz. The subjects were instructed to keep their heads still during the stimulus interval and then point their noses towards the perceived location and press the response button. The position was recorded by the head tracker and stored to disk. A sample of the results is shown in Fig. 6A.

The experiment was repeated, this time allowing the subjects to turn their heads during the stimulus interval. As shown in Fig. 6B the subject performs better than in the stationary head condition. Since subjects can move their heads to position the virtual source in front of them, the task can now be considered a centering task.

The data shown here are meant to be a proof of concept, and are by no means complete. The results shown in Fig. 6 used the ITD model as the spatialization method. The performance using the HRTF model is very similar to the results using the ITD model.

6.2. Effects of System Latency

A preliminary experiment to judge the effects of system latency shows the versatility of the system as well as demonstrates the need for small latencies. In this experiment the azimuth of a virtual Gaussian noise source was moved in a reproducible manner. The subjects were instructed to follow the noise by keeping their noses pointed in the direction where they perceived the source. A system latency was artificially induced by buffering the head tracker by various amounts. The stimulus rendering included reading input and position data from disk, reading head position from the head tracker through the serial port, filtering with an HRTF chosen using the egocentric source position, and streaming head position data to disk for storage. Figure 7A shows a trial in which a subject tracks the sound source. The two trajectories are aligned and the difference in the paths is evaluated. The error was separated into overshoot and undershoot error based on the sign of the error and whether the source trajectory was concave up or concave down. This error metric allows us to consider error due to the acceleration of the source. Changing the induced system latency produces differences in source tracking performance. Figure 7B shows that there is a significant difference in the overshoot error at 32.2 ms.

This preliminary data shows that there may be interesting effects for latencies smaller than those previously considered relevant. Previous studies using statically located stimuli only found statistically significant differences in performance when latencies exceeded 70 ms [3]. The difference in results illustrates that the effect of system latency is dependent on the task being performed.

7. CONCLUSION

The virtual display system described here will support psychophysical experiments that including moving sources, multiple sources, and head movement. The system has met our goals of being flexible and easy to use for users with different skill sets. Although the models of spatialization still need to be explored, the system has proven to be accurate and stable in realizing these models. The ITD spatialization model has worked accurately and reliably. The HRTF model gives a proper spatialization but suffers from issues of spatial undersampling that need to be resolved independent of the system. The advantages of this system are:

- System is versatile and easily usable by psychoacousticians with typical skill sets.
  - Matlab interface allows use of implemented models and easy experimental design without knowledge of how models are implemented.

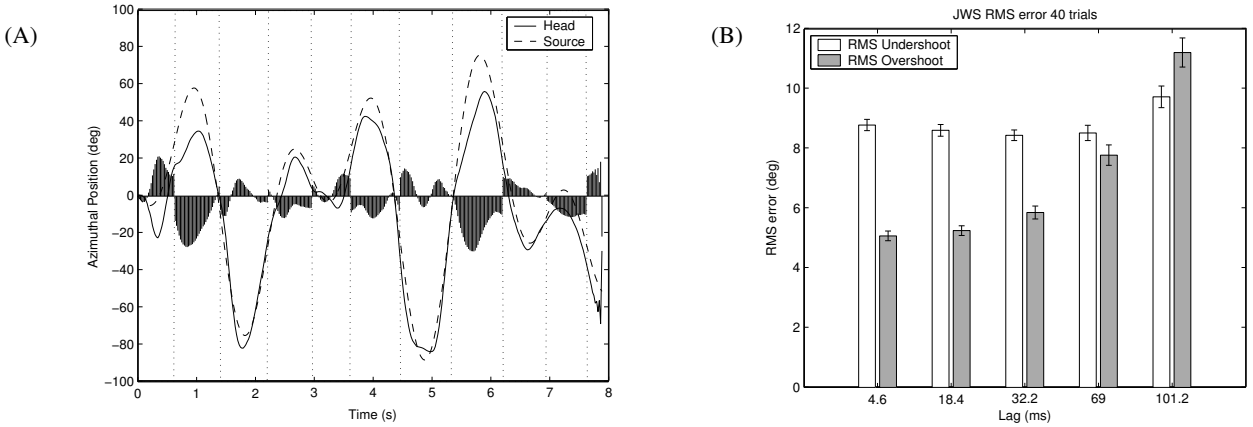

Figure 7: Sample subject data for tracking experiment. a) Example of a trial. The dashed line is the path of the rendered sound, the solid line is the path of the subject's head *corrected for best delay*. Shaded areas represent undershoot (negative) and overshoot (positive) error. b) RMS error of subject JWS for 40 trials at different induced tracker lags.

  - Models are easily created and adapted without knowledge of system architecture.

- Object-oriented approach allows for intuitive multi-source experimental design.
- Ability to easily stream data to/from hard drive allows for arbitrary input signals and position vectors, as well as for recording of head movements.
- Low System Latency < 7 ms which can be improved to a theoretical lag of 23 μs and negligible time jitter with improvements in tracker technology.
- The effects of small system latencies on human performance need further investigation.
- System uses standard hardware and has modular software components allowing for easy hardware upgrades.
- System is scalable allowing for more complicated algorithms when processor speed is increased.
- System is open source and freely distributable, allowing for others to easily implement and adapt to meet specialized goals.

Acknowledgments: The authors would like to thank Jonathan Bettencourt for his guidance and technical support. We would also like to thank Abhi Kulkarni and Barb Shinn-Cunningham for their guidance as well as Tom Metkus, Vijay Ullal, and Osonde Osoba for their work in development and testing. This work was partially supported by NIH NIDCD DC00100 and NIH RO1 NS34425.

8. REFERENCES

[1] E. Wenzel, "Effect of increasing system latency on localization of virtual sounds," in *Proc. AES 16th Int. Conf. on Spatial Sound Reproduction*, 1999, pp. 42–50.

[2] F. L. Wightman and D. J. Kistler, "Resolution of front-back ambiguity in spatial hearing by listener and source movement," *J. Acoust. Soc. Am.*, vol. 105, no. 5, pp. 2841–53, 1999.

[3] D. S. Brungart, B. D. Simpson, R. L. McKinley, J. A. Kordik, R. C. Dallman, and D. A. Ovenshire, "The interaction between head-tracker latency, source duration, and response time in the localization of virtual sound sources," in *Proceedings of the tenth meeting of the international conference on auditory display*, July 2004.

[4] J. W. Scarpaci and H. S. Colburn, "A real-time virtual auditory system for spatially dynamic perception research," *J. Acoust. Soc. Am.*, vol. 115, no. 5, pp. 2599, May 2004.

[5] A. D. Dorval, D. J. Christini, and J. A. White, "Real-time linux dynamic clamp: A fast and flexible way to construct virtual ion channels in living cells," *Ann. Biomed. Eng.*, vol. 29, pp. 897–907, 2001.

[6] V. R. Algazi, R. O. Duda, D. M. Thompson, and C. Avendano, "The CIPIC HRTF database," in *Proc. 2001 ieee workshop on applications of signal processing to audio and electroacoustics,*, 2001, pp. 99–102.

[7] D. J. Kistler and F. L. Wightman, "A model of head-related transfer functions based on principal components analysis and minimum-phase reconstruction.," *J. Acoust. Soc. Am.*, vol. 91, no. 3, pp. 1637–47, March 1992.

[8] Z. A. Constan and W. M. Hartmann, "On the detection of dispersion in the head-related transfer function," *J. Acoust. Soc. Am.*, vol. 114, no. 2, pp. 998–1008, 2003.

[9] G. F. Kuhn, "Model for the interaural time difference in the azimuthal plane," *J. Acoust. Soc. Am.*, vol. 82, no. 1, pp. 157–167, July 1977.

[10] A. W. Mills, "On the minimum audible angle," *J. Acoust. Soc. Am.*, vol. 30, no. 4, pp. 237–246, April 1958.

[11] R. M. Hershkowitz and N. I. Durlach, "Interaural time and amplitude JNDs for a 500-hz tone," *J. Acoust. Soc. Am.*, vol. 46, no. 6, pp. 1464–7, Dec 1969.

[12] T. I. Laakso, V. Valimaki, M. Karjalainen, and U. K. Laine, "Splitting the unit delay," *IEEE Signal Proc. Mag.*, vol. 13, pp. 30–60, 1996.

[13] Intersense, "IS-900 Precision Motion Tracker Manual for Models IS-900 VET, IS-900 VWT & SimTracker," Tech. Rep., 2003, Doc. No. 072-00060-0F00 Revision 2.1.