Radio drum gesture detection system using only sticks, antenna and computer with audio interface

Ben Nevile
University of Victoria / Cycling '74
bbn@saoul.ca

Peter Driessen
University of Victoria
peter@ece.uvic.ca

W. A. Schloss
University of Victoria
aschloss@finearts.uvic.ca

Abstract

We present the results of our ongoing project researching a tighter coupling between computer and performer. The audio-input radio drum is presented, a simplification of the original apparatus that provides superior latency and resolution. Different demodulation schemes for the amplitude modulated input signals are discussed. Techniques to analyze gesture data are outlined, including eestimation of parameters and detection of events. In the case of the latter, a statistical basis for making decisions in the presence of noise is presented. The algorithm used to detect when the audio-input radio drum has been hit is outlined.

1 Introduction

For many years computer technology has been used by musicians to shape and synthesize sound, but to actually *play* a computer with the dynamic feel of an acoustic instrument remains difficult. At the root of the problem is the human-computer interface: we need interfaces that feature very low latency to enable the rapid sonic feedback loop that a musician enjoys with an acoustic instrument. We strive to satisfy the 10ms low latency criterion establi shed for satisfactory reactive performance systems (Wessel and Wright 2002), and also want to minimize the amount of variation or slop in the timing of our instrument. We want an interface with enough dynamic range to allow expressive performances, and the technology employed must be stable and predictable.

In *Communicating with Meaningless Numbers* (Zicarelli 1991) David Zicarelli points out that the performer-instrument communication is better characterized by a continuous engagement of control rather than by a few discrete events. Zicarelli argues that to find better methods of controlling sound synthesis we ought to be transmitting data that is stripped of all musical context. The stream of these "meaningless" numbers can then be analyzed and interpreted in software.

This paper presents the results of our ongoing project researching methods to create this flow of meaningless numbers between our computers and the gestural interface known as the *radio drum*. Our approach is to transcode the gesture data into the **signal** domain. In other words, we have chosen to communicate, manipulate and analyze the gesture data with the same isochronous mechanisms that we use to manipulate audio data. This creates the tightest possible coupling between input and output.

Several gesture input systems have been evaluated, including CNMAT's Connectivity Processor (Rimas Avizienis 2000), Electrotap's Teabox (Allison and Place 2004), and our own amplitude modulation toolkit. As we became more familiar with the technology in use in the radio drum, it became apparent that there was a simpler solution that's more efficient, less latent, and more precise.

Next we'll outline the apparatus and methods employed to communicate the gesture data from the radio drum to the computer. Following that we'll discuss the analysis of signal data, first generally and then in the specific context of the radio drum. The relationship between events in the signal and message domains is discussed, and we'll present our solution to the problem of how to detect when the surface of the drum has been hit.

2 The Audio-Input Radio Drum

The idea of gesture sensing using capacitive moments was first developed at Bell Labs by R. Boie (Boie, Ruedisueli, and Wagner). The original apparatus is outlined in figure 1: a special control box drives the two drum sticks with frequencies in the 50-55kHz range. This driving potential induces the movement of charge on the four surfaces of the antenna. These four signals are input into the control box, which demodulates the frequencies into DC signal levels that it can then analyze to determine the three-dimensional position of each stick, as well as determine when the drum has been hit. Data is output from the control box using the MIDI protocol: each of x, y, z and velocity in the case of a hit are reported as 7-bit MIDI messages. The x, y and z positions are sent to the

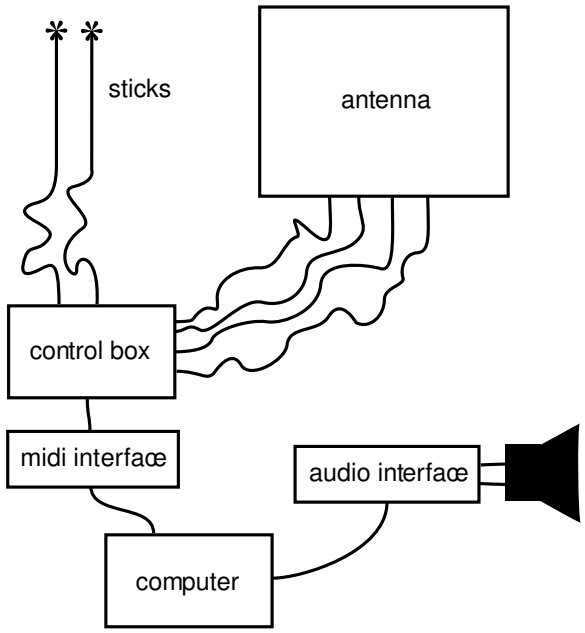

Figure 1: Old apparatus for the radio drum

computer 50 times a second.

The apparatus for our audio-input radio drum is outlined in Figure 2. The driving signals sent to the sticks are generated by the audio outputs of a multi-channel sound device. The outputs of the antenna are connected to the inputs of the same multi-channel sound device. All interpretation of the signals is handled in software. The control box is no longer necessary, and MIDI is no longer needed.

In this audio input scheme the computer both generates the driving signals and receives the signals produced by the induced voltage changes in the antennae. Being a physical, electrodynamic system with scale much shorter than the wavelength of the radiation, the transmission time from the sticks to the antenna is virtually instantaneous. The latency of the system is therefore exactly the round-trip latency of the computer and audio device. With a professional quality sound card this latency can be less than 5 milliseconds.

2.1 Carrier Frequencies

Using this new system we are free to choose the transmitting frequencies that we send to the sticks. The two frequencies must be spaced far enough apart so that the sidebands created by the movement of the sticks do not spectrally overlap; figure 3 illustrates this idea. The spectral width of each sideband is dependent on the physical nature of the interface; in the case of our drum, it is reasonable to use 450 Hz, the upper bound on the frequency of muscular contraction (Nakra

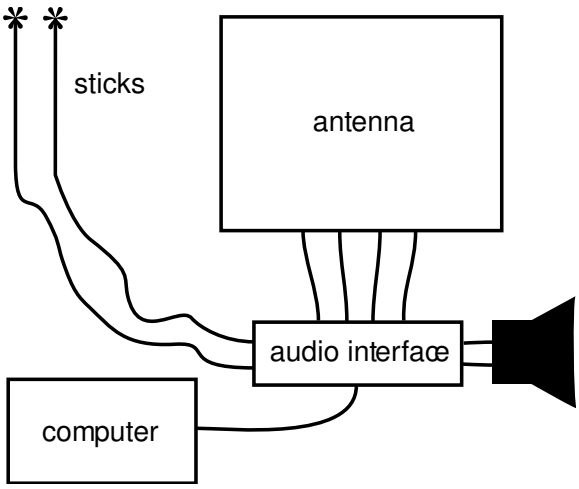

Figure 2: Apparatus for the audio-input radio drum

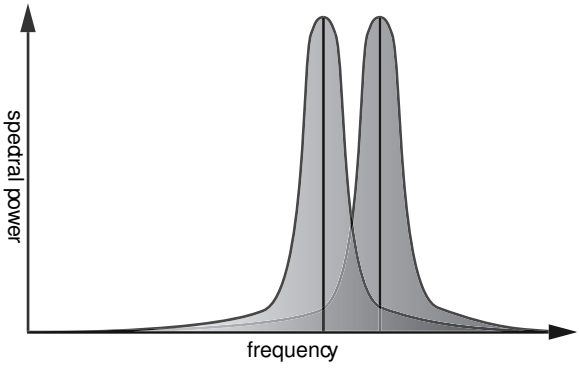

Figure 3: Illustration of the input spectrum. Two carrier frequencies must be chosen far enough apart so that their bandwidths do not significantly overlap. Bandpass filters are used to isolate the carriers from one another.

2000). Note that when the stick makes contact with a solid, unforgiving surface, high frequency components can be the result. This could cause a problem when separating the two channels of data, but we have not found this to be a problem in practice.

We are of course limited to frequencies less than half of the audio output device's maximum sampling rate, which on a modern device can be as high as 192kHz. Figure 4 shows a plot of the antenna's receiving power as a function of the driving frequency. The vertical axis of the plot is scaled logarithmically with base two and normalized relative to the maximum receiving power around 60kHz. Therefore one can think of the value of the vertical axis as the number of bits lost relative to the maximally efficient frequency. Another fac-

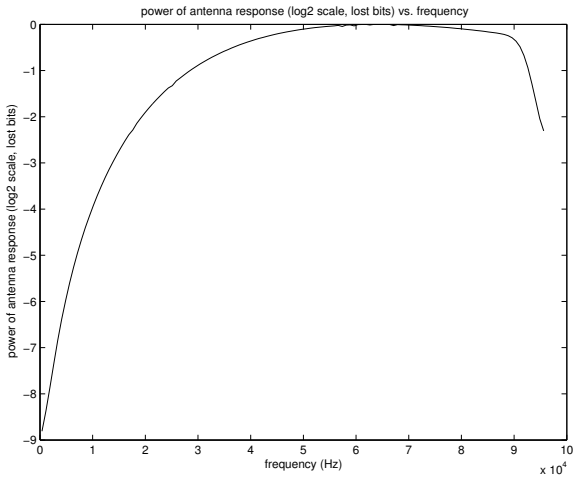

Figure 4: Receiving power of the antenna versus the frequency of the carrier. Note that the vertical axis is logarithmically scaled.

tor in choosing carrier frequencies is that operating a signal-processing environment at high frequency is computationally demanding. This is not because of the amount of calculating involved, but rather primarily due to the amount of data that needs to be moved through the computer's IO subsystems. In our performances we have found we achieve acceptable performance characteristics when operating the audio device with a sampling rate of 64kHz and generating carrier frequencies of 26 and 30kHz.

2.2 Demodulation

Each input signal contains the two modulated carriers superimposed, so filters are needed to seperate them. The ideal box filter for a given carrier would be centered at the frequency of that carrier, and would be just wide enough to accomodate all of the gesture-generated sidelobes without attenuation. In practice we have found that biquadratic bandpasses of the kind intended to be used for audio are adequate for the job, although it's likely that custom designed filters could yield more efficient results.

AM demodulation techniques that can be used when the original carrier is available are called *synchronous*. Synchronous demodulation has better signal-to-noise characteristics than asynchronous demodulation (Haykin 1994). Since in our audio-input interface system the carrier is generated in software, we can demodulate synchronously. A standard synchronous demodulation technique is to multiply the amplitude-modulate signal by the original carrier wave. If the carrier is operating at a frequency ω and the amplitude envelope of the modulated carrier is $A(t)$, the amplitude modulated carrier would

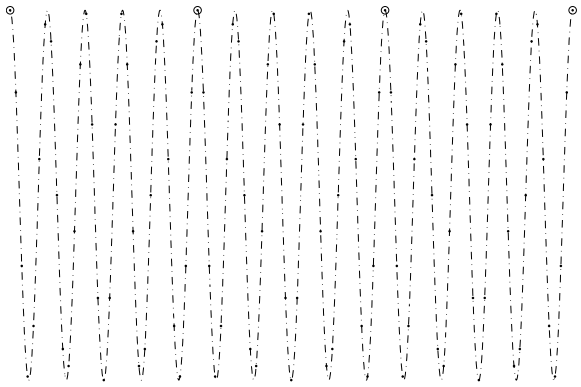

Figure 5: Illustration of downsampling technique. The carrier frequencies are set so that an integer number of periods fit exactly within the downsampled sample rate. In this case exactly five periods of the sine wave fit within thirty-two samples at the main sampling rate. The dots indicate the points that the sine wave is sampled at the main sampling rate, and the larger circles indicate the points at which the downsampling mechanism chooses a sample.

be $A(t)\cos(\omega t)$. The multiplication of this function with our own carrier for synchronous demodulation can then be expressed as

$$A(t)\cos(\omega t)\cos(\omega t+\phi) = \frac{A(t)}{2}(\cos(2\omega t+\phi)+\cos(\phi)) \quad (1)$$

Note that we have added a phase shift of ϕ in the multiplying carrier since in general the two carriers won't be perfectly in phase. The envelope $A(t)$ that we want to extract is now modulating a sine wave at twice the frequency of the original carrier, but it also is sitting in the baseband. By high-pass filtering out the doubled carrier frequency component of the signal we are left simply with $A(t)\cos(\phi)/2$.

Figure 5 illustrates a more computationally efficient method of moving the amplitude envelope into the baseband that could be called *synchronous downsampling*. In this scheme a naive downsampling of the signal is performed at a rate that exactly matches an integer number of periods of the carrier signal. The carrier frequency is shifted to zero, and what was previously a bandpass filter becomes a lowpass filter. When operating at a sample rate of ω_r and downsampling by a factor of n, the carrier frequencies must be $k(\omega_r/n)$ where integer k satisfies $0 < k < (n/2) - 1$. So for example at a sampling rate of 64kHz and downsampling by a factor of 32, we can use carrier of $2k$ kHz, where $1 <= k <= 15$. In Figure 5, $k = 5$. Carriers of 30kHz and 26kHz ($k = 15$ and 13) have worked well for us.

In both demodulation methods, the amplitude of the demodulated information is dependent on the phase of the input carrier wave. Since we are generating these driving frequencies it's important to tune their phase to maximize the signal-to-noise ratio of our instrument. In a noiseless environment the bit depth of our audio interface, the signal strength transmitted by the sticks, and the gain of the antenna are the factors that limit the signal-to-noise ratio.

3 Gesture Data Analysis

The remainder of this paper discusses strategies to deal with the inevitable noise picked up by the antenna. That is, we can model the jth sample of an input signal x as

$$x_j = s_j + n_j \tag{2}$$

where s_j is the true value of the sampled signal and n_j represents the corrupting noise. In the modern world there are many sources of electromagnetic interference that our antenna will pick up; we have found a realtime spectral analysis of the input signal to be a very valuable tool to aid in identifying noise from surprising sources such as light fixtures, televisions, and refridgerators. If the noise cannot be stopped at the source, the corruption can be minimized by setting the carrier frequencies so that the overlap between the gesture sidelobes and the noise is minimized.

When the sources of noise are small and many, we can thank the Central Limit Theorem for allowing us to make the reasonable generalization of treating the background noise as *white* - that is, having constant power per unit of spectral bandwidth over the spectral range of interest **??**. This simplifies our analysis considerably when we consider our two main analysis problems: first, how to use the $s + n$ data we collect to optimize our determination of the true value s, and second, how to determine when an event has taken place. In Electrical Engineering textbooks these are called the problems of *estimation* and *detection*.

3.1 Parameter Estimation

Consider a *locally stationary* signal - that is, one whose true value we expect to be unchanging over a certain period of time. Given only a single sample of data x_0 from this signal, no separation of the true value and the noise is possible. Accordingly the best estimate of the true value of the signal is simply x_0 under the assumption of zero-mean white noise. When multiple samples $x_0, x_1, x_2, ...$ are taken into consideration we can exploit signal processing techniques to provide a more accurate estimation of the true value of the signal s. The most straightforward way to reduce the effect of zero-mean white noise in a stationary signal is to take the mean of m successive samples. If the variance of our gaussian noise is σ_n^2, the variance of the estimate will be σ_n^2/m (Schwarz and Shaw 1975).

Typically the signals we'll be estimating will not be stationary. If our sampling rate is high compared to the bandwidth of the gesture data, it is not unreasonable to assume *locally linear* signal movement for a small number of samples. Therefore we can still take a mean of m samples, but we'll be estimating the value of the signal in the middle of those m samples. In other words, the averaging of a signal over time introduces latency of $(m-1)/2$ samples into the calculation of the estimate.

This intuitive argument can be backed up with filter theory - averaging m samples after all is just a crude FIR lowpass filter, and if the sampling rate is very high our gesture data will be relatively low in the spectrum. It is generally preferable to use one of the many design techniques to design a more optimal IIR or FIR filter with a response curve that is flatter through the important baseband and then drops off quickly to attenuate the higher frequencies.

More complicated estimation algorithms that account for non-stationary signals include the Kalman filter (Robert Grover Brown 1992), a recursive estimator that can take into account multi-dimensional correlation and provides the smallest linear mean-squared error, and non-linear algorithms which are more effective for non-gaussian noise sources but require information about the statistical distribution of the noise and the signal (Schwarz and Shaw 1975). In all of these schemes a filter that increases the signal-to-noise ratio comes at a cost of increased latency in the estimation.

For the Boie drum the parameters of interest are the x, y and z positions of each stick. The geometry of the four segments of the antenna is such that the signal strength for each is associated with one of the surface's four edges. The z position of a stick is estimated simply by summing all four of its signals. The x and y positions are estimated by calculating the ratio of the signal strengths of the two edges perpendicular to the dimension in interest. The resulting parameters are very close to linear near the center of the drum pad, but become non-linear near the edges. More importantly for event-detection purposes, our parameters are monotonic.

3.2 Events in Signals

The method by which events are represented in the signal world can be confusing for those of us used to dealing with message-based events in a computer music programming environment like Max. Consider an *event signal* whose value represented the status of an event: a value of 1 indicates the event has occurred at that sample time, and a value of 0 means it has not. At audio sampling rates, most event signals would

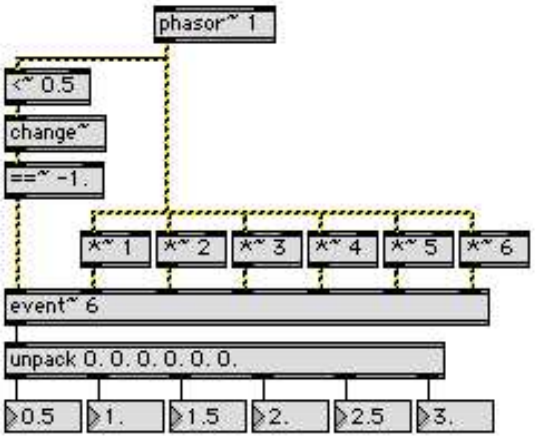

Figure 6: Max patch featuring an example of an event capturing MSP network. The phasor object generates a signal that ramps from 0 to 1. When it crosses 0.5 the output of the ¡ object changes from 1 to 0, which causes the change object to output a -1. The -1 causes the == object to output a 1, which triggers a sampling of the six input signals by the event object. The next time the high-priority scheduler thread is serviced the event object outputs the values of the sampled parameters as a single list.

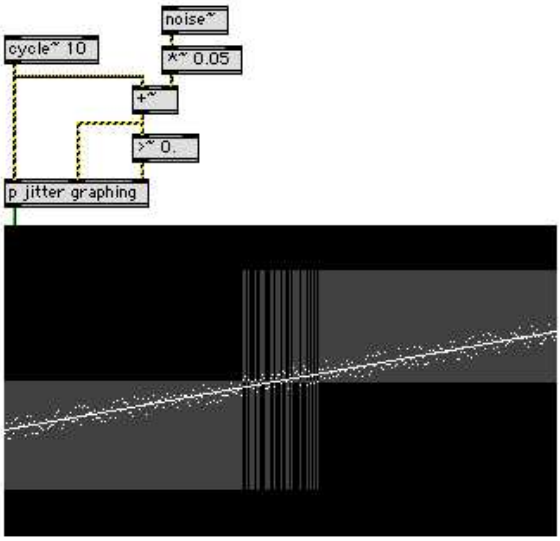

Figure 7: Max patch featuring an MSP network that executes a naive test and uses Jitter to graph the results. Noise is added to a sine wave to imitate a noisy signal. This signal is tested to see if it's greater than zero. True and false results are indicated by the background grey bar graph.

be made up mostly of zeroes with the occasional one. Accordingly, it is inefficient to process for each and every sample the calculations that might happen as a result of an event occurring.

In the Max/MSP environment the solution is to move into the message-passing domain. The `edge~` object sends bangs out of its first and second outlets in response to zero to non-zero and non-zero to zero transitions in the input signal, respectively. But beyond simply knowing that an event has happened, we needed the ability to synchronously sample values from multiple parameters at the time of an event, as well as queue multiple events in a single audio processing vector to be output into Max's high-priority scheduler thread. So the `event~` object was created; Figure 6 shows an example of how this object can be used to move between the signal and message domains. Also noteworthy in this example is the `change~` object, which is frequently useful when an event is to be detected based on a change in the value of a signal.

Max's high-priority scheduler thread can operate in several modes (Jones and Nevile 2005). In overdrive mode with the scheduler in audio interrupt enabled the queued events will be output and processed before the next signal vector is calculated. Therefore there is a latency introduced in this process whose delay in samples can be modeled as a random variable with uniform probability distributed over the range from zero to the number of samples in the signal vector size. Operating at a signal vector size of 16 with a sampling rate of 64kHz The maximum latency introduced here is one quarter of a millisecond.

3.3 Tests and Statistics

Given one or more streams of signal data, we have to create tests to help us determine when an event has taken place. The corruption of the signals with noise requires that we closely evaluate the performance of all testing algorithms.

Consider Figure 7, which shows a Max patch that features an MSP network that executes a simple test and uses Jitter to graph the results. Noise is added to a sine wave to imitate a noisy signal, and the signal is tested to see if it's greater than zero. True and false results are indicated by the background grey bar graph. As it approaches the zero threshold, the noise around the signal rapidly moves the combined value above and below the zero threshold, causing the result of the test to flip-flop many times. To make decisions in the presence of noise obviously will require methods more sophisticated than this naive approach.

As illustrated in Figure 8, the `thresh~` object allows us to set a hysterisis range that can eliminate the rapid flip-flopping. If the input signal moves from below to above the object's upper limit parameter, the output signal is set to 1. If the input signal drops from above to below the object's lower

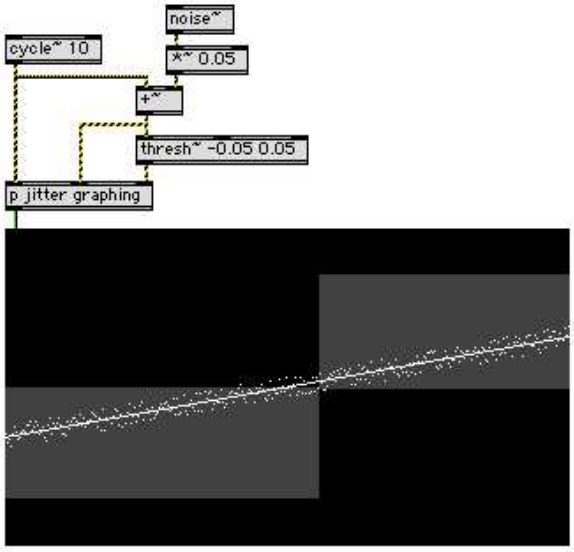

Figure 8: The same network as in figure 7 but with a thresh object instead of a naive $>$ object.

limit parameter, the output signal is set to 0.

Another standard building block of analog logic is the envelope follower, which tracks changes in a signal's direction of motion and reports the extremities reached. Figure 9 shows how one might naively implement an envelope follower with MSP objects. The system fails because the noise in the signal causes the difference between successive samples to alternate quickly between negative and positive values. To attack this problem we created the `fuzzyenv~` object, which can be seen in Figure 10. `fuzzyenv~` takes a distance parameter that is the amount the signal must reverse from its extreme value before a change in direction is reported.

Note that in both of these examples the noise could be quantified precisely - ie, there would never be more than 0.05 added to or subtracted from the signal. In the real world signals are random variables that if we're lucky can be characterized as gaussian variables with a known variance. We therefore cannot create tests that are guaranteed to give us the right result, but we can tune the parameters of the tests to distribute the risk in a way we find acceptable. In any statistical test there are two types of failures: a *false alarm* is when the test indicates that an event has occurred when in fact it has not, and a *miss* is when a test does not detect an event that has in fact occurred. It is unfortunately impossible to simultaneously optimize our tests to minimize both of these possible failures. There is a subtle balance that must be struck between the probability of correct detection, the false alarm rate, and the miss rate. Furthermore, latency (or detection time) is another factor in this tradeoff, since better estimation techniques

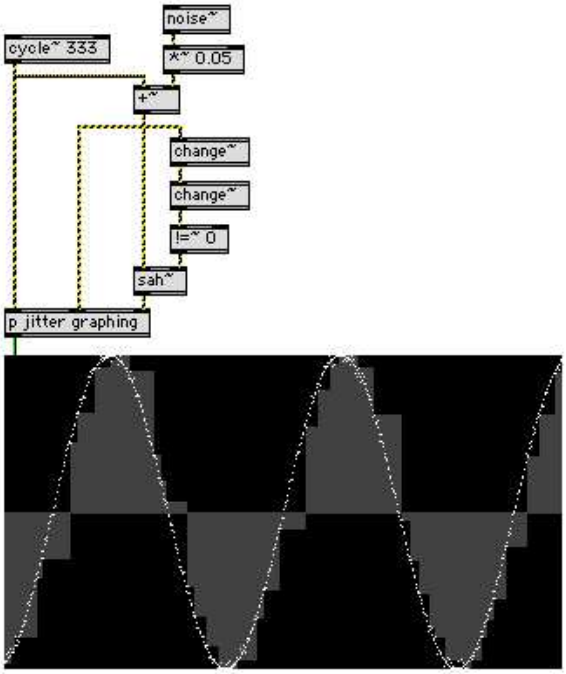

Figure 9: Max patch featuring an MSP network that implements a naive envelope follower and uses Jitter to graph the results. Noise is added to a sine wave to imitate a noisy signal. The envelope of the signal is tracked using two change objects in series followed by a sah to sample-and-hold the envelope extremeties.

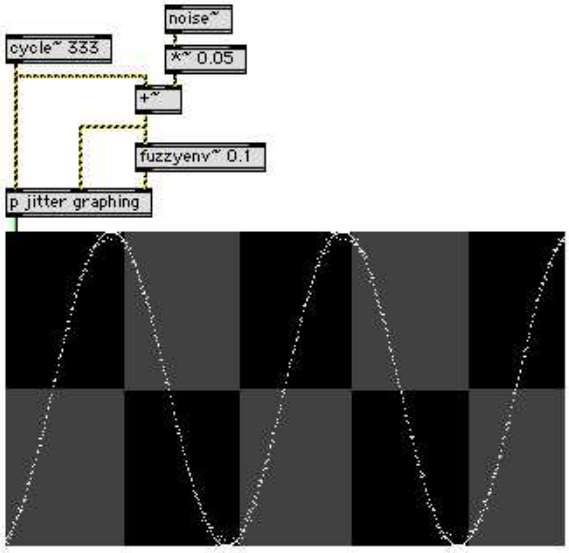

Figure 10: The same network as in figure 9 but with a fuzzyenv object instead of a naive envelope follower.

that reduce the uncertainty in the signal values take longer to perform.

If the noise in a signal is gaussian and we know the variance, it is straightforward to evaluate the probability of the various errors in a particular scenario using standard statistical tables or an approximation formula for the area under the standard normal curve (Wald 1947). It is difficult, however, to get a broad idea of how often a test will fail, since one can only calculate the probability of failure in a very particular situation. In theory one can calculate not only the probability of failure, but the average time between failures and even the distribution of the time interval between failures.

When the conditions that define the occurence of an event have been identified we can tailor a test procedure to our statistical comfort levels. In situations where there is more than one condition that defines the event, it's important to keep in mind that the different estimation and testing algorithms can introduce different latencies into the signals. When features are to be identified in parallel, care must be taken to ensure that the respective latencies inherent in the processing are of equal length.

3.4 Audio-Input Drum Events

The algorithm developed to detect when the surface of the drum has been hit with one of the sticks has three conditions, one for each of the position, velocity, and acceleration of the signal. Each test requires information about the signal, so an automated calibration procedure has been developed. The Java code employed records a few seconds of data with the sticks placed on the surface of the drum, and then a few more seconds with a performer hitting the drum forcefully and moving the sticks through their entire range of motion. The code then analyzes the recorded signals to distribute scaling factors that normalize the four incoming signals. The ranges of the estimated x, y and z positions of the sticks are also normalized based on the data collected, as is the velocity for detected hits. The variance of each estimated parameter is also measured to be used to set thresholds in the tests discussed below.

The position test ensures that the stick is positioned very close to the surface that is being hit. Since our problem reduces to motion in the z dimension, our test simply ensures that our z estimate is below a threshold. The value of this threshold is the mean z value calculated during the calibration procedure plus a factor of the calculated standard deviation in the z position estimate.

The velocity test ensures that the stick is moving downwards towards the surface at the time of the hit. The z-velocity of the signal is estimated as the difference between successive z-position estimates. In other words,

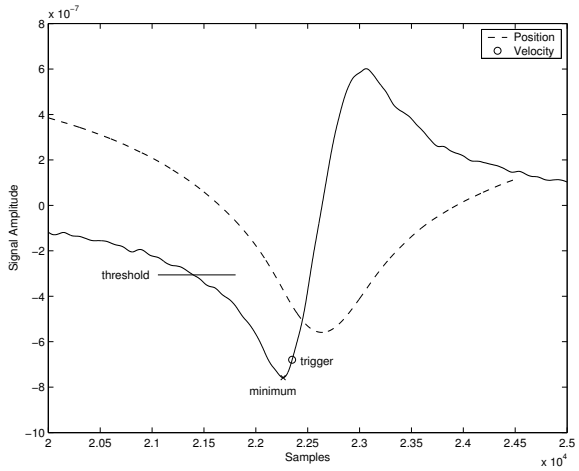

Figure 11: Position and Velocity Signals of a Radio Drum Hit

$$
\begin{aligned}
v_j &= z_j - z_{j-1} && (3)\\
&= (s_j + n_j) - (s_{j-1} + n_{j-1}) && (4)\\
&= (s_j - s_{j-1}) + (n_j - n_{j-1}) && (5)
\end{aligned}
$$

Unfortunately although it looks like the noise components of the signal subtract from one another, in fact the variance of this estimate of the velocity is twice the variance of the z-position estimate (Schwarz and Shaw 1975). Our test simply compares the estimated velocity with a threshold whose value is set as a factor of the standard deviation of the estimated velocity signal calculated during the calibration procedure.

The acceleration test ensures that the downward velocity has reached a minimum value and is experiencing a restoring force upwards. This is implemented with a `fuzzyenv~` object on the velocity estimate - note that the total change in velocity is simply the *integration* of the acceleration! The variance in the difference between two velocity estimates spaced m samples apart is twice the variance of the velocity estimate and four times the variance of the z position estimate. The reversal threshold of the `fuzzyenv~` object is set as a factor of two times the standard deviation of the velocity estimate calculated during the calibration procedure.

Figure 11 shows a plot of the position signal corresponding to a drum hit and its velocity estimate on the same graph. One thing to note is that the minimum of the velocity happens *before* that of the z position estimate; the difference between this point and the minimum of the z-position signal is approximately 7 milliseconds. Figure 12 shows a simplified version of the MSP network used to implement the hit detection algorithm. The `event~` object captures three parameters in association with every hit: the velocity, which is sent as the

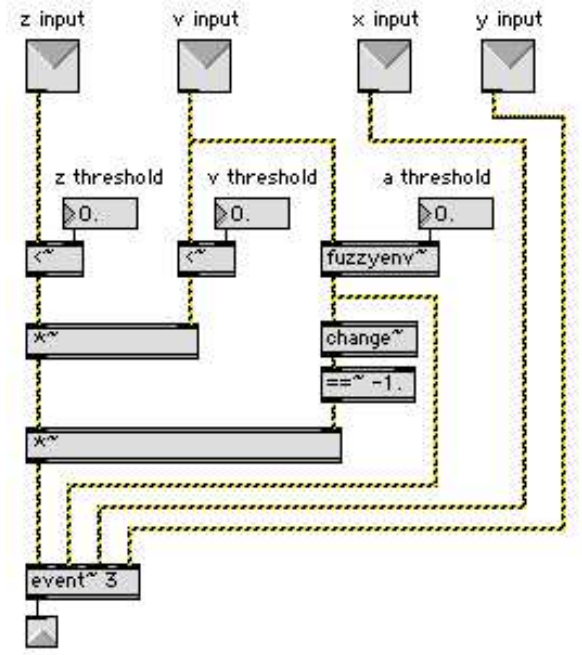

Figure 12: Simplified version of the MSP network that implements the hit detection algorithm.

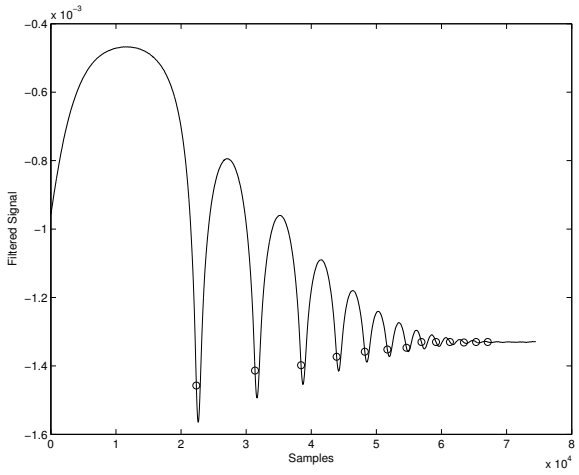

Figure 13: Filtered Signal With Detected Hits

minimum extrema from the `fuzzyenv~` object, and the estimated x and y location of the hit. Figure 13 shows a plot of a recorded signal of a drum roll and the hits detected by the algorithm.

4 Conclusions

We have developed a new gesture capture system that requires less hardware, is highly resolved both temporally and quantitatively, and boasts extremely low latency. On a modern laptop computer the processing required to demodulate the signals and detect hit events is less than five percent of the signal processing power available. The latency between a hit and the resulting sound synthesis is dependent on the round-trip latency of the sound device and operating system - in our case this is a little less than six milliseconds. In a typical environment the signal-to-noise ratio of the position signals is more than 85dB; the signal-to-noise ratio of the velocity parameter of a hit is more than 70dB. Most important, the system is solid and reliable and has been used in several concerts by different musicians across North America this year.

There is much to do in the future to make this interface even better. Different frequency transmitting systems and estimation algorithms may improve the signal to noise ratio. Linearizing the cartesian parameters of the drum will allow a more intuitive programming interaction when setting up mappings. Different antenna geometry may improve the three-dimensional tracking and signal-to-noise ratio of the instrument. Finally, we are excited about the possibility of interacting with physical models with a controller capable of transmitting high-resolution physical details.

References

Allison, J. T. and T. A. Place (2004). Teabox: A sensor data interface system. *Proceedings of the 2004 International Computer Music Conference*.

Boie, R., L. W. Ruedisueli, and E. R. Wagner. Gesture sensing via capacitive moments. internal memo at AT&T Bell Labs.

Haykin, S. (1994). *Communication Systems, 3rd ed.* United States: John Wiley and Sons, Inc.

Jones, R. and B. Nevile (2005). Creating visual music in jitter: Approaches and techniques.

Nakra, T. M. (2000). Searching for meaning in gestural data. *Trends in Gestural Control of Music*.

Rimas Avizienis, Adrian Freed, T. S. . D. W. (2000). Scalable connectivity processor for computer music performance systems.

Robert Grover Brown, P. Y. H. (1992). *Introduction to Random Signals and Applied Kalman Filtering*. New York: John Wiley & Sons, Inc.

Schwarz, M. and L. Shaw (1975). *Signal Processing: Discrete Spectral Analysis, Detection and Estimation*. Dubuque, Iowa: McGraw Hill.

Wald, A. (1947). *Sequential Analysis*. New York: John Wiley & Sons, Inc.

Wessel, D. and M. Wright (2002). Problems and prospects for intimate musical control of computers. *Computer Music Journal*.

Zicarelli, D. (1991). Communicating with meaningless numbers.
Computer Music Journal 15(4), 74–77.