Contact and Free-Gesture Tracking for Large Interactive Surfaces

by

Che King Leo

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

Author ....
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by ....
Joseph A. Paradiso, Ph.D
Principal Research Scientist, MIT Media Lab
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Contact and Free-Gesture Tracking for Large Interactive Surfaces

by

Che King Leo

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2002, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

In this thesis, I redesigned a Responsive Window that tracked the locations of physical impacts on a pane of glass. To do this, I redesigned algorithms and boards, and utilized a DSP to do the signal processing. Tracking accuracy was significantly improved over the previous system. A study on radar rangefinding systems was also done to select the best radar system for integration with the Responsive Window. The applications for the system were realized, and debuted with great success.

Thesis Supervisor: Joseph A. Paradiso, Ph.D
Title: Principal Research Scientist, MIT Media Lab

Acknowledgments

To **Joe Paradiso** for all his insight, intuition, and confidence in me. I thank him for giving me the chance to work for him. It has been a great, rewarding experience.
To **Stacy Morris** for all her help and insights, as well her willingness to help review the previous incarnation of this thesis.
To **Mats Nordlund** for his insight and help with the Radar.
To **Steve Francis** for his insight and help with the DSP.
To **Axel Yamamoto** for his timely resourcefulness with Panasonic transducers.
To **Mark Feldmeier**, **Josh Lifton**, **Hong Ma**, and **Nick Yu** and all my colleagues and friends in the **Responsive Environments Group** for putting up with me and helping me when I needed help.
To all my **friends** for their patience, support and encouragement.
Finally, to my **family** for their encouragement, love and support. Without them, I would never have gotten this far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Description

The quest to create interactive rooms has inspired much interest by the research community [1]. Indeed, many groups across the US work towards the greater goal of one day being able to live in a “smart” house fully facilitated by the use of computers and the latest technologies. As an example, there are advantages in making sensors which can be easily retrofitted to surfaces already available in homes easily retrofitted with sensors to make them interactive. Many windows have already been retrofitted with anti-theft systems. After all, the more information a computer has about its environment, the better it can aid the user. In this thesis we discuss methods for creating an interactive system from already in-place surfaces. This system would be able to generally monitor the user’s position and activity within a given area, and feature a large, accurate, tracking surface for detailed interaction.

1.2 Prior Work

In 1998, Ishii and Paradiso collaborated to create a project called Ping Pong Plus (PP+) [2]. In this project, a projector cast images onto a Ping Pong table that were correlated to where the ball had bounced on the table, allowing the players to attain a higher level of engagement. The tracking mechanism used in PP+ consisted of distributed microphones, and used a simple time of flight analysis to determine the position of the bounce. In this way, the ping-pong table was made into an interactive surface. Naturally, this straightforward technique could be expanded to other continuous surfaces, most notably glass.

The use of glass as an interactive surface is appealing in many ways. First and foremost, it is one of the most common elements in society today. Windows are in most any man-made structure, allowing many places to be easily retrofitted with an interactive surface. Second, glass is usually clear. Given sufficient throw, a projector and a screen can be placed behind the glass to allow images to appear, while never having parts of the image blocked by users, a problem found in most user interfaces that use a projector. In 2000, Paradiso *et al.* [3] proposed and demonstrated transforming large sheets of ordinary glass into an interactive surface capable of tracking the location of knocks via acoustic pickups adhering to the glass.

In 2001, Checka presented a more robust system of easily retrofitting pieces of glass into interactive surfaces [4]. Using piezoelectric polyvinylidene fluoride (PVDF) strips to pick up flexural bending waves created by users knocking on the glass, the system was able to roughly track and process the user input. The system was demonstrated by converting an ordinary conference room wall into a fully interactive web browser. However, there were a few potential drawbacks to this approach that prevented it from practical deployments. Most notably, the use of the interpretive language MATLAB and a commercial A/D PCI card to process the knocks caused considerable “dead time” of the system, and therefore limited the user’s rate of input. Also, the use of a 3rd-order coordinate calibration model caused some points to be more significantly inaccurate than others. Finally, the system was limited to knocks made by human knuckles. This system was incapable of differentiating many different types of inputs, such as the difference between higher frequency metallic knocks (via metallic objects like rings and keys) and more conventional human knuckle-based knocks. This is crucial for tracking knock position because the wavefronts from these types of knocks travel at very different velocities because of the dispersive nature of the glass. This system was also unable to correctly discern valid knocks which produce reliable coordinates from poor knocks that produce unreliable coordinates and false triggers caused by external sources such as pickup from cellular phones and sharp sounds like loud, nearby hand clapping or door slams. Finally, the system was only interactive in two dimensions, lacking the ability to have depth perception. Such an ability would seem to be a very useful feature, since it would provide an additional dimension's worth of information for the system to use. The screen, for instance, could respond differently as users approach.

1.3 General Overview

In this thesis, the Responsive Window system presented above was modified so that it became faster, more robust, more accurate, and produced more features. Changes ranged from rewriting the tracking and signal processing algorithms to modifying the hardware used in the system. In addition, a second system of sensor input was investigated: a low-power range-finding radar, which would allow for a third dimension depth of detection. The addition of a system could add robustness to the interactive window system by monitoring nearby human presence, providing an entirely new level of interaction.

The system described by Checka [4] has been modified in several ways. First, the piezoelectric PVDF foil strips were replaced by piezoceramic microphones (Panasonic EFVRT series) which are less noise-sensitive [5]. These sensors were glued to the glass and covered by a shield box, reducing the total amount of electromagnetic pickup noise introduced into the system. An Analog Devices digital signal process on (DSP) replaced the National Instruments Card with MATLAB for data acquisition and the differential time-of-flight analysis. Figure 1-1 shows the current hardware configuration for the system.

Two extra microphones, labeled "bash" and "clap" were used to provide additional information to the system. The bash microphone and circuitry which consisted of an electrodynamic microphone with a gain stage and lowpass filter was used to cleanly detect low frequency "bashes"(under 300 Hz; i.e. hitting the glass with the palm of the hand), since the piezo-electric pickups had a poor frequency response for low frequencies. The "clap" microphone was used to veto ambient noise from the surrounding off-window environment that falsely triggered the system. It was mounted in the air, instead of on the glass. Its analog signal conditioning involved several gain stages, a peak detector and a bandpass filter. By detecting when a large amplitude existed on the clap veto microphone, the system was able to use this information to reliably veto most of the false triggers to the system.

The analysis algorithms were substantially changed for improved performance and real time operation. A heuristically-guided chi-squared estimation was used to select which portions of the waveform most closely matched each other, so that those portions could be cross-correlated to fine-tune the accuracy of the values used for time-of-flight analysis. An iterative algorithm implicitly solved the problem of finding the intersection of the two hyperbolas defined by the time-of-flight values. This was done by matching the differential time-of-flight values by minimizing the error over the x-coordinate and the y-coordinate separately via a logarithmic search. The algorithm also counts the number of zero-crossings in the waveforms over a certain period of time. This simple frequency analysis was used in conjunction with the time-of-flight values in a nearest neighbors algorithm [6] to determine the dominant frequency of the flexural wave, thereby providing the correct speed of the waveform, a necessary value for correct calculation of the knock position.

In order to make the system aware of a third dimension, it will be augmented with a simple, low-power range-finding radar that looks through the surface of the

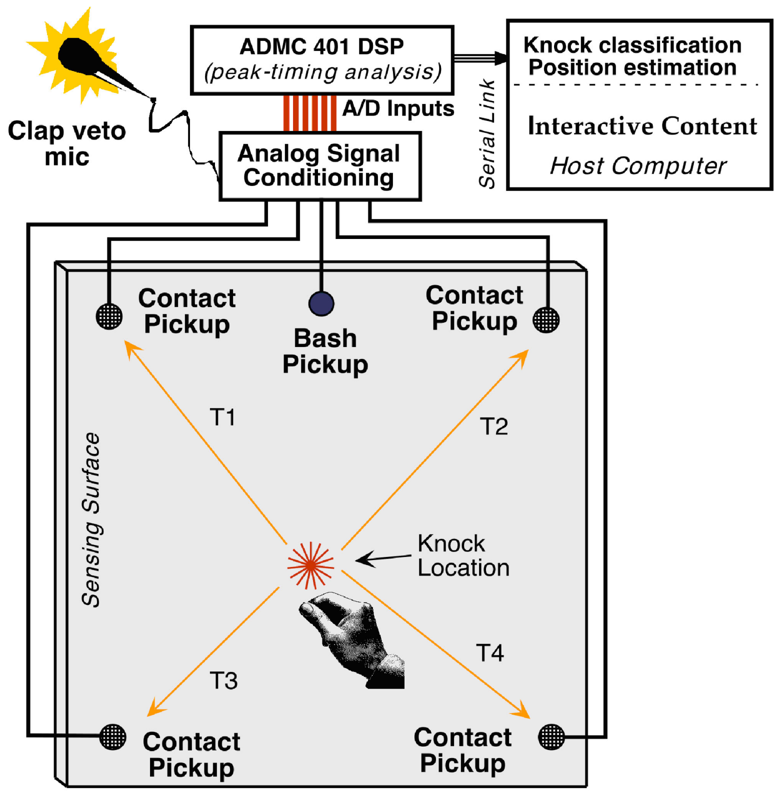

Figure 1-1: Current hardware configuration for Responsive Window system.

glass. The radar will be monitored by the computer that drives the media content and video projector. This device also assisted the algorithm in several ways. First, in conjunction with the clap microphone, it reinforced the system to prevent false triggers, by instructing the system not to process a knock event if there is no user standing in front of the window. A second benefit from the range-finding radar was its ability to track and zone users as they moved about the window. Such information allowed the display to adjust its projection, evoking the feeling of interaction beyond knocks on a piece of glass, thus allowing users to fully immerse themselves in the window.

Chapter 2

Background and Motivation

2.1 The Previous Responsive Window Prototype

The previous system developed by Checka [4] described a system that employed polyvinylidene fluoride (PVDF) as contact sensors to track acoustic impacts on the window. Strips of PVDF backed with foam were mounted on a small PCB board that amplified the acoustic signals to increase the signal to noise ratio. This is shown in shown in Figure 2-1. Tiny nuts were then precisely super glued to the surface of the glass at desired locations and used to screw mount the board. The PVDF strips were thus tightly pressed against the glass and were able to pick up flexural waves originating from user input.

The acoustic waveforms created by knocks on the glass were amplified by the PCB board and sent through analog signal conditioning to remove high frequency components. After this, the signal was digitized by a National Instruments NI-6024E Data Acquisition Card and processed by MATLAB on a PC to discern the position of the acoustic impact on the glass. The algorithms used to derive position consisted of a third-order fit based on time-of-flight information obtained using cross-correlation analysis. The third order fit was based on data gathered via calibration. Calibration consisted of knocking on twenty known positions on the glass, and calculating the fit

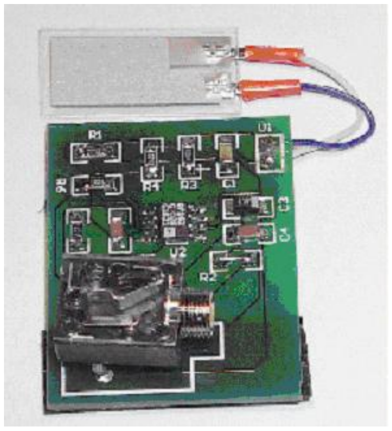

Figure 2-1: Original PVDF with pickup

over these twenty variables.

When using the third order model, the system achieved an accuracy of less than a 5 cm standard deviation from the knock positions, when the knocks occurred on the calibration points. Checka also described a method for calculating time of flight from a simple rising edge analysis, which produced a 10 cm standard deviation from the knock position.

2.2 Motivation

In the quest for a system that can be responsive to user input, we wanted to create a system that responded to acoustic impacts in real-time, as well as to a wider spectrum of user input, unconfined to the rigors of meticulously setting up a precise calibration system. We also desired a system that improved on accuracy and/or precision. Lastly, we wished to create a more robust system, which was less likely to be fooled by fake input (i.e., ambient noise and other false triggers), while still being responsive in some way to all types of valid user input.

2.2.1 Real-Time Response

The use of MATLAB as the signal processing tool, was beneficial for developing algorithms, due to its close-to-infinite precision arithmetic. However, depending on MATLAB running on a PC is a poor choice to employ in creating a system that needs to process information at a rate better than the maximum rate of human knocking (5-10 Hz). In the previous system, using the NI-6024E and MATLAB setup incurred approximately one to two seconds of delay before a response was shown on the screen. To improve on this delay, some of the algorithms were implemented on a digital signal processor (DSP), as described in this thesis.

2.2.2 Increased User Input Spectrum

The previous system was only able to track knocks created by an impact with one knuckle. This was because the calibration knocks used to calculate the coefficients for the forward determination scheme in the third order fit were all knocks using a single knuckle on each calibration point of the glass. Using multiple knuckles or different sections of the hand, including the palm, the fist, and fingernails often led to inconsistent results; thus the previous system could only determine knock position caused by knocks made with the knuckle. The new system was designed to be able to track impacts caused by other sources, such as metallic taps from rings and keys. The system was also able to track and/or respond to a wider range of knuckle knocks. This is vital, as people will knock in any fashion in a public venue, and one cannot reasonably expect them to be highly constrained.

2.2.3 System Robustness

The previous system ran on Windows 98, and the machine required rebooting every day. Without rebooting once a day, continual knock estimation would fail, because the NI-6024E and MATLAB combination would stall and stop processing taps, causing the system to freeze. A migration to a digital signal processor (DSP) for signal discretization and part of the signal processing easily improved system robustness by eliminating the signal processing done on the PC. Second, the sensors would also pick up random noise before the amplification stage. This problem was solved by changing the sensors and placing a shield of copper on the pre-amplifier board. Finally, the PVDF sensors would fall off easily, as the strength of their attachment to the glass relied upon the screw connection between the pre-amplification PCB boards and the nuts glued to the glass. By changing the type of sensors to a cheap, commercially produced device (piezoceramic telephone receivers, found to be superior in performance to the other pickups tested for this purpose), it allowed the sensors to be super-glued to the glass, thereby eliminating the mechanical instability of Checka's "pinched" sensors.

2.2.4 System Portability

The previous system also did not lend itself to portability. The requirement to have a PC with an open PCI slot for the NI-6024E card, as well as needing prototype analog electronics did not allow the system to be easily brought from place to place for display. The complicated setup and extensive requirements for different software made the system less portable. It was also desirable to build a system that could be eventually manufactured cheaply (at a cost of less than 100 per unit). The use of the $3000 NI-6024E data acquisition card prohibited this requirement. Therefore, a DSP was used for signal discretization and initial signal processing.

2.2.5 Self Calibration

Another requirement for the new system was that it should have an easy to use user interface (UI) that would allow any user to calibrate the Responsive Window system. Because every sheet of glass could potentially have different characteristics, a program was needed that was versatile and allowed the user to easily calibrate the system.

2.2.6 Free Gesture Tracking

To allow for free gesture tracking, radar rangefinding systems were investigated for integration into the Responsive Window. This integration will allow the system to react not only to two dimensional input on the sheet of glass, but also the third dimension of depth away from the surface. Doing so broadened the spectrum of applications for the system.

Chapter 3

Responsive Window System Design

3.1 Design Overview

This section describes the Responsive Window system in four parts: Hardware Design, DSP Software Design, PC Software Design, and Window Calibration.

The hardware design for the Responsive Window consisted of a redesign of the analog signal conditioning board, as well as a design of the digital components, and the physical implementation of the system (i.e. where the sensors are placed on the window, and other physical structures used to improve robustness). The PVDF strips were replaced with Panasonic piezoelectric receivers. Two more sensors were introduced to the system - a crystal microphone for vetoing high energy noise that can falsely trigger the system, and an electrodynamic microphone for detecting low frequency "bash" impacts that did not get picked up well by the Panasonic receivers. The pre-amplication boards were designed to increase bandwidth, to prevent accidental destruction of delicate analog parts, and to make them modular. The DSP chosen to do the signal processing and signal discretization was the Analog Devices ADMC401 DSP chip, which has the ability to interface with a computer via a serial cable link.

The software design had three main parts: DSP Code, PC Knock Position Deter-

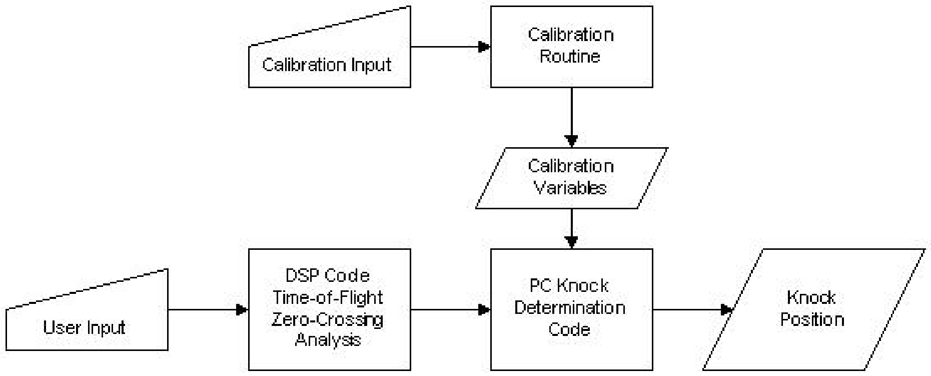

Figure 3-1: Flow diagram for determining knock position

mination Code, and Calibration Code. The Calibration Code calculated parameters to be used by the PC Knock Position Determination Code. The entire knock position determination software design for the Responsive Window required both DSP and PC code. The DSP code extracted useful characteristics for determining knock type, as well as time-of-flight information between two predetermined pairs of sensors. The PC knock determination program, was written in C^{++}, and used parameters obtained from calibration code and parameters from the DSP to calculate the knock position. Figure 3-1 shows the flow diagram for the Responsive Window knock position determination.

3.2 Hardware Design

The hardware redesign was done by using a bottom-up approach. First, the appropriate sensors were chosen to replace the PVDF strips. Then, the pre-amplifier board was redesigned. Next, the analog signal conditioning board was redesigned to include channels for all the microphones. Then, a DSP was chosen, and a board was made to power different sections of the hardware, followed by physical enclosures for the system. Finally, sensor placement on the window was considered.

3.2.1 Replacing the PVDF Strips

First, different types of microphones were investigated to replace the PVDF strips, since the PVDF strips did not have uniform frequency responses. Also, the mounting for the PVDF strips allowed various amounts of background noise to enter the system, depending on how tightly strips were pressed against the window. Therefore, manufactured transducers were investigated, including microphones, speakers and other sensors. The final choice was a Panasonic piezoelectric cellular phone receiver (EFVRT series). The receiver had a 300-5000 Hz range for its frequency response, which covered a portion of the frequencies of the knocks of interest. This Panasonic receiver had the best performance across all types of knocks compared to all other transducers tested for this application. A low frequency impact "bash" microphone was used to complement the Panasonic receiver in the very low frequency range, where bashes produced the most response. However, the Panasonic receivers were susceptible to interference from cellular phone transmission signals when cellular phones which used the GSM standard were very close (e.g. a few feet), so this flaw was compensated for in the DSP code.

3.2.2 Low Frequency Impact Microphone

To allow the system to respond to low frequency "bash" impacts (such as those from the fist of a hand), a fifth microphone was used. An electrodynamic microphone was chosen because it responded well at lower frequencies. In order to be maximally responsive in the lower frequency range, the microphone had to be glued with only its diaphragm stuck to the window. Thus, in order to allow the microphone to operate in a stable manner (without falling off), we had to epoxy the center of the diaphragm to the window, and cantilever the mass of the microphone while strain-relieving the cable. Figure 3-2 depicts the electrodynamic microphone with its diaphragm glued to

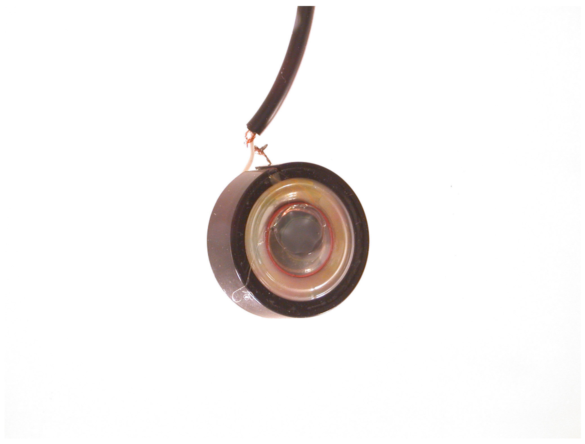

Figure 3-2: “Bash” microphone glued to the window surface

the surface of the glass. The resultant assembly was resonant at the low frequencies excited by a bash. This microphone was conditioned by a circuit that used simple analog peak detection to determine whether or not very low frequencies had been detected. Figure B-3 in Appendix B diagrams the final circuit used.

The peak-detected signals from the low frequency impact microphone were used to determine if a “bash” event had occurred that was beyond the frequency response range of our piezoelectric receivers.

3.2.3 External Noise Transient Veto Microphone

The quest to allow for a broad user input spectrum also allowed the system to be susceptible to external events that could falsely trigger an event. The main source of false triggers came from high-energy transient signals, such as hand clapping. The energy of hand clapping can couple into the glass and cause the receivers to falsely pick them up as knocks. Therefore, we placed a sixth microphone in the air so that the system could tell if the signal was stronger in the air or in the glass, thereby indicating the origin of the trigger. A simple crystal microphone was used that had a very nice frequency response to the types of sharp, transient signals that created false triggers.

Analog components were designed to properly filter the signals from this transducer. Figure B-3 diagrams the final circuit. A high gain stage was provided for the signal, since an amplifier board was not used for this microphone, and out-of-band noise was filtered.

3.2.4 Redesigned Pre-Amplifier Boards

In order to minimize induced noise and electromagnetic pickup, the first stage of the preamplifier was mounted right at the transducer. The original pre-amplifier boards [4] were built with two goals in mind. The first was to allow the boards to be screwed onto nuts crazy-glued onto the surface of the window, so that the sensors could be removed without damage. The second was to improve the signal to noise ratio (SNR), by amplifying the signal before it was transmitted to the main board.

A 3-conductor cable using a mini-stereophone jack coupled the power, ground, and output signal to the preamplifier boards. When the stereophone cable was physically plugged into the pre-amplifier board jack, there was a brief moment when the signal out was shorted to ground. The previous setup thus allowed full power to be dumped out of the output of the preamplifier IC, an OP162, into ground. Hence, if this was done while power was on, it would sometimes result in the OP162 burning up. In order to limit the output current, we placed a small resistor (330 Ω) was placed in series with the output to the OP162.

Also, since the piezoelectric microphones required a high impedance input, R4 and R5 were increased to 2.0 MΩ in order to get a better signal from the output. Finally, in order to provide modularity, the board was designed so that it could be used for microphones that required either a low or high impedance front end. Figure B-1 from Appendix B shows the new schematic for the pre-amplifier board.

The area board was decreased by 45%, so that the amount of stress and torque on the sensor was reduced. The use of the smaller board size allowed for the use of

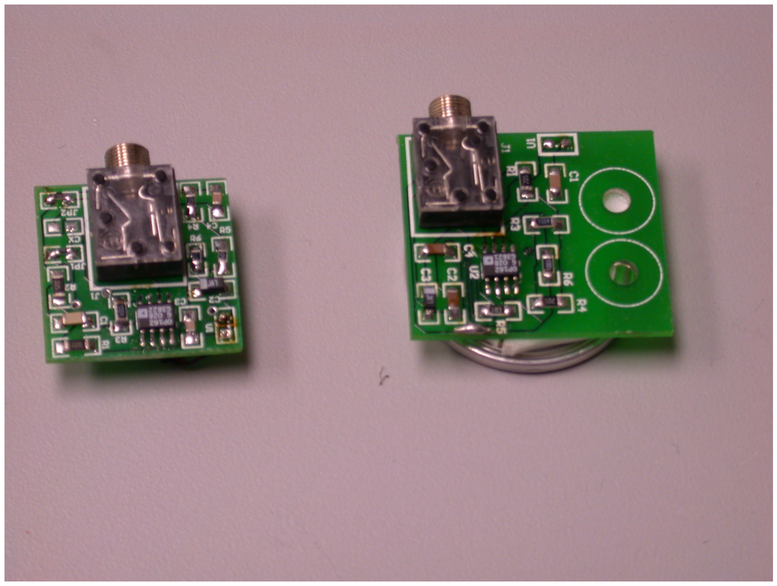

Figure 3-3: Comparison of size between redesigned (left) and previous pre-amplifier board (right).

a wider range of sensors, (i.e., sensors with smaller footprints). Figure 3-3 compares the sizes of the two boards.

To reduce the amount of noise pickup at the preamplifier stage, a physical shield made from copper siding material was fitted on top of the preamplifier board. Figure 3-4 depicts a board fitted with the copper shield.

3.2.5 Digital Signal Processor

The digital signal processor (DSP) that was chosen had to meet several requirements. First, the DSP had to be able to sample on at least six channels, allowing it to sample the inputs from the four contact microphones, plus the two other microphones. Second, the DSP needed a sample rate well above the Nyquist frequency. The Panasonic EFVRT receivers had a frequency range up to 5 kHz, so to properly sample the wavefront signal, the DSP had to have a minimum sampling throughput of 50 kHz per channel. Third, a good data resolution of at least 10 bits was desired. Finally,

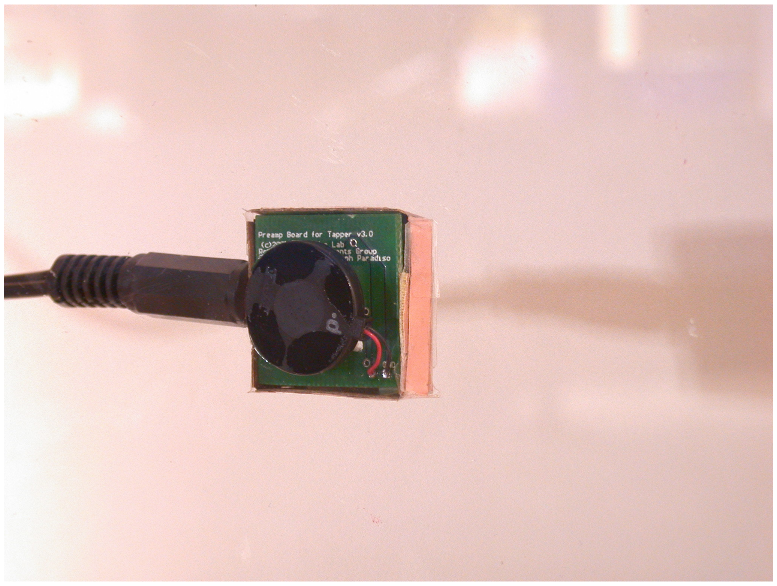

Figure 3-4: Preamplifier board with copper shielding

the DSP needed to have a serial output connection to allow the DSP unit to easily interface with most computers, either via a USB serial port or a legacy serial port.

At the time of the DSP selection, the only readily available solution that met all of these requirements was an evaluation kit from Analog Devices - the ADMC401 Motor Controller Kit. The ADMC401 kit had eight channels of inputs, a sampling rate of 50 kHz, 12-bit resolution, and a serial port interface.

3.2.6 Sensor Window Placement

The four main contact sensors were placed so that each sensor was not too close to the window edge, to minimize the effects of reflections from the closest window edge which could contaminate the wavefront arrival. Because the highest dominant frequency traveled at speeds ranging from 1200 m/s to 3000 m/s (depending on glass thickness), a 50 kHz DSP sampling rate was insufficient to allow for sensor placement in arrays. this would have allowed the position to be determined via a linear array of sensors with phase calculations. The best solution for our constraints was to place

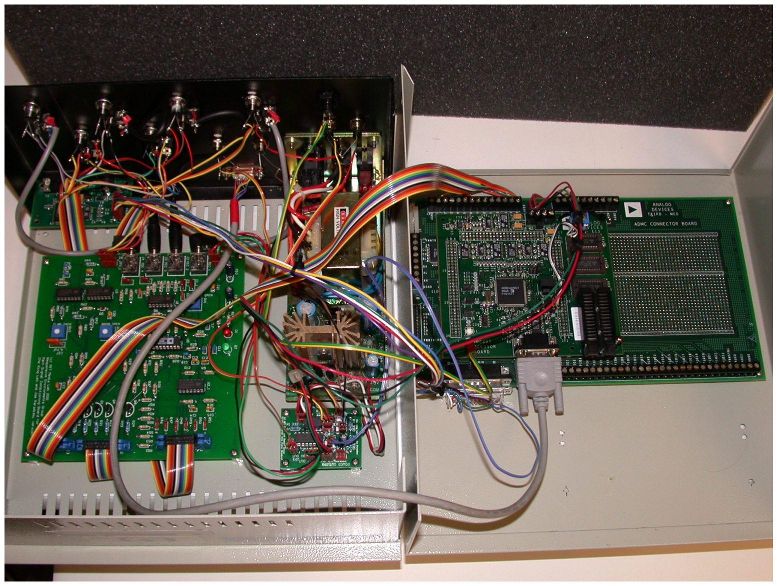

Figure 3-5: Internal view of housed electronics

each sensor some constant distance from each corner. Chapter 5 discusses experiences with different sensor placements on the window and their implications.

The “bash” microphone was placed at the top center of the window. The electrodynamic microphone needed to be able to sense very low frequency flexural waves from its location. The “clap” microphone was placed in front, concealed out of view at the top of the display. This was so that users would not see the microphone, and deliberately try to trick the system into falsely triggering.

3.2.7 Physical Hardware Implementation

To protect the electronic hardware in the system, all the analog and digital components were placed inside a sturdy $11.5'' \times 8.5'' \times 5.5''$ aluminum box. Power was provided to the system by a $\pm 5V$ power supply. A breakout board was used to distribute various voltages required by system components, including $9V$ for the serial driver, $5V$, and $-5V$ for the analog and digital electronics. Figures 3-5 and 3-6 depict the internal and external view of the box.

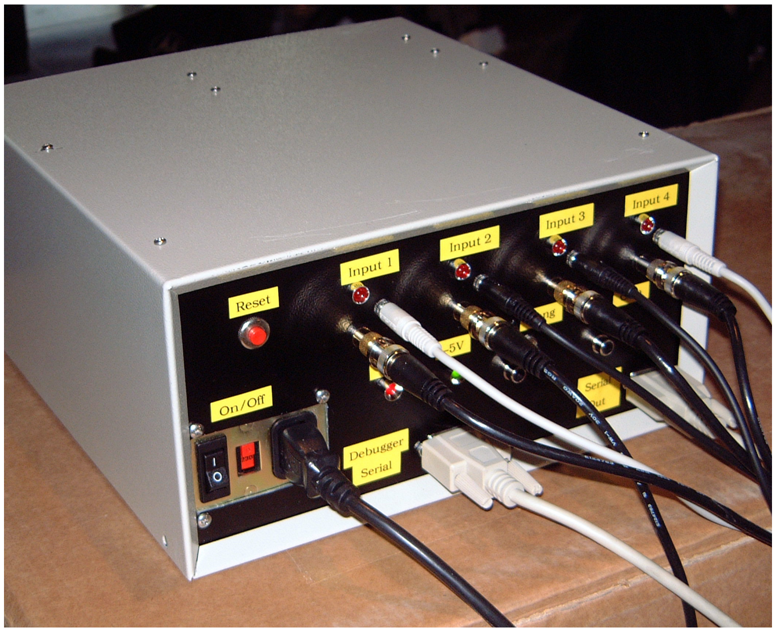

Figure 3-6: External view of housed electronics

3.3 DSP Software Design

The DSP Code was used to extract features from the sampled waveforms, and send these features to the PC Knock Determination Program. The DSP code can be broken up into several parts - waveform normalization, trigger threshold, cellular phone interference detection, zero crossing count, heuristic waveform selection, rising edge selection, and analysis of the signals provided by the “bash” and “clap” microphones. Figure 3-7 is a flowchart for the DSP software.

A dynamic trigger threshold on the window-mounted sensors controlled the point at which samples from the sensors would start to be processed. Next, any vetoes were derived to stop processing for false triggers, such as cellular phone noise. Events considered valid were then further processed to acquire a zero crossing count, and a heuristic waveform selection and rising edge analysis was performed to find differential time-of-flight parameters.

Before the DSP software design is discussed, it is informative to investigate the

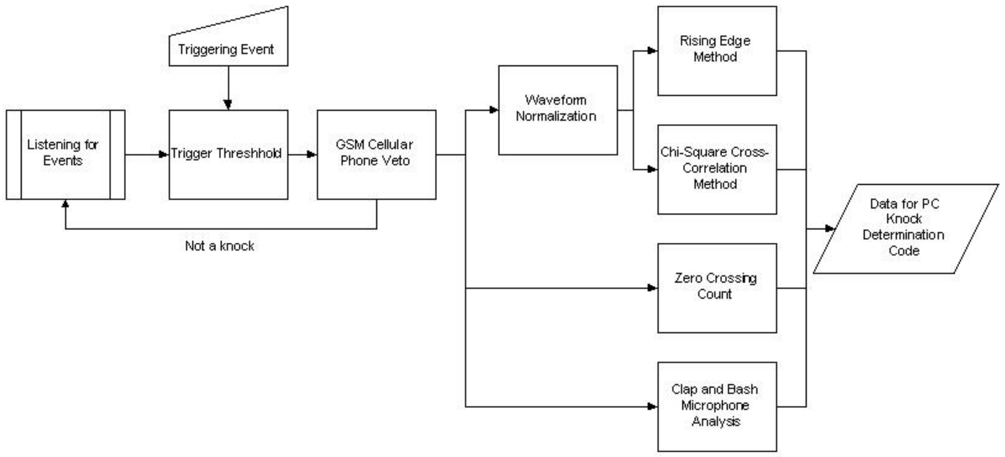

Figure 3-7: Flowchart for DSP software

characteristics of the waveforms sampled by the DSP.

3.3.1 Sampled Waveform Analysis

Figure 3-8 illustrates several types of the waveforms that can be obtained from various impacts on a 143 cm $\times$143 cm, $\frac{1}{4}$ inch thick tempered glass, mounted on a wooden frame. As the metallic taps produced a much larger response in the transducers, they tended to saturate more (although the most relevant initial wavefront remains linear.)

Hard, "metallic" impacts were found to excite higher frequencies than most knuckle impacts. This is evident not only from looking at the Discrete Fourier Transform (DFT) of the sampled signals (with the DC gain subtracted out), but also from hearing the timbre of the sound made from the knock. Figure 3-9 shows the magnitude of the frequency response for the various waveforms shown in Figure 3-8.

There are two big peaks at around the vicinity of 1000 Hz for knuckle-based impacts, and one peak at 3200 Hz for metal-based impacts. Regardless of objects used for impact, it seemed that the only major frequencies that could be excited by users for this particular sheet of glass were located around 1000 Hz and 3200 Hz. Because of the

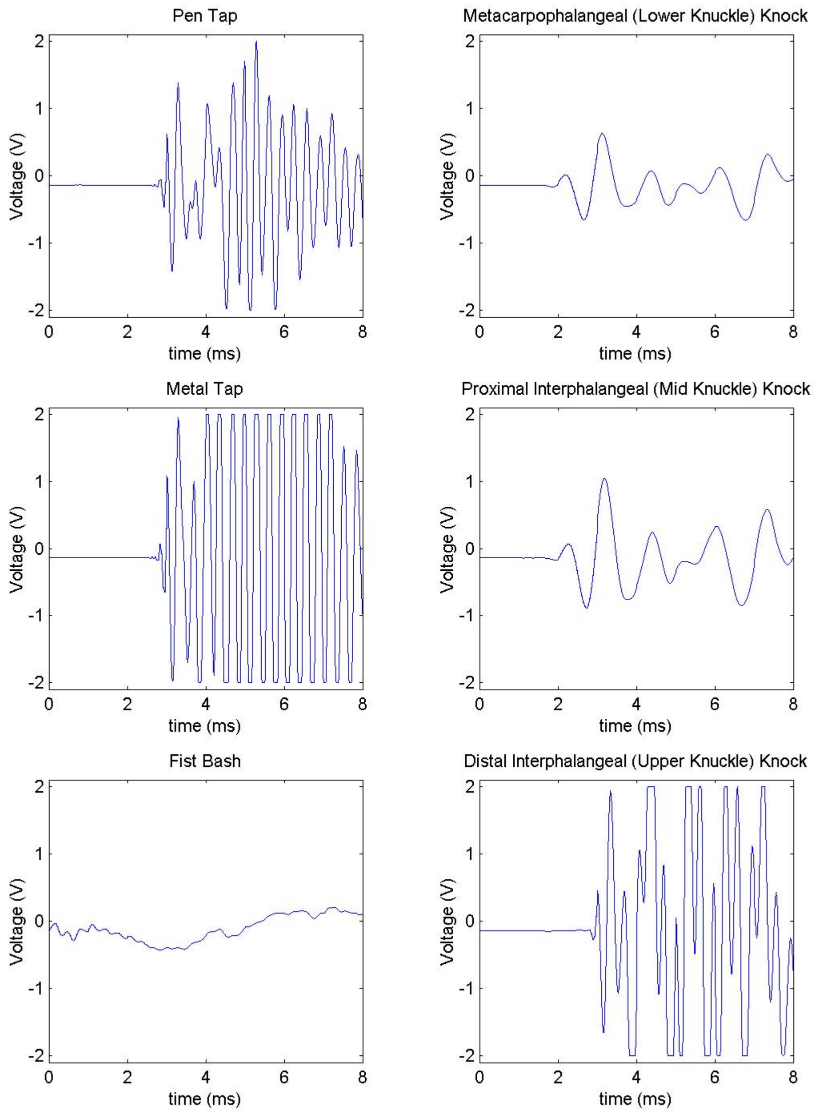

Figure 3-8: Sampled waveforms for different tap types

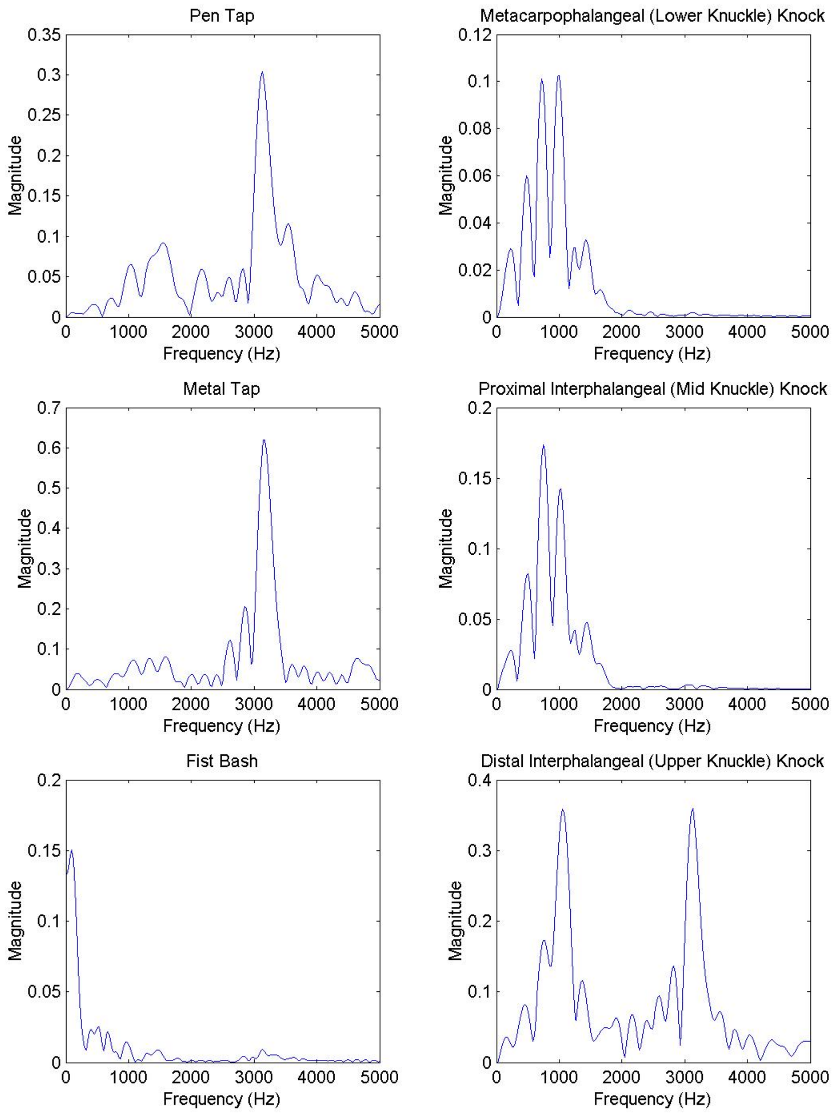

Figure 3-9: DFT of sampled waveforms for different tap types

dispersive nature of the glass, different frequency components in the glass propagated through the glass at different speeds. Thus, calculating the velocity for both dominant frequencies became essential for determining knock position. Because the two peaks at 1000 Hz traveled at relatively the same speed, the average speed for knocks exciting that frequency range was used. Perplexingly, the distal interphalangeal joint (upper knuckle) knock has frequency content in both areas. In order to properly calculate the knock origin for the upper knuckle knock, a velocity associated to the frequency at 1000 Hz was used. This led to an algorithm for determining the difference between knuckle and metal knocks.

If the mean of each set of waveform samples was brought to zero by subtracting out initial bias, and took the absolute value of each waveform, “lobes” that represented the flexural waves caused by the impact were left. At some point the peaks of each lobe no longer increase in magnitude and the waveform appears to become “random.” This “randomness” is mainly caused by the dispersive nature of glass, the nature of the impact excitation, and the reflections from the glass edges as the glass lapses into modal oscillations. Thus, in order to accurately find a good differential time-of-flight between two sensors, it would be easier to only compare “clean” sections of the signal where only one frequency component is dominant. This is a departure from the cross-correlation method that Checka [4] used. In the previous system, Checka performed cross-correlation on entire 8 ms samples of both signals. Here, this system seeks to increase precision by correlating specific sections of each sample which were more clearly from one dominant frequency.

3.3.2 Dynamic Trigger Threshold

In order to process knock events, it was important to know when the timing and the number of input signal samples were valid for processing. At first glance, this would seem easily determined by taking some amount of samples before and after a set threshold was reached; however, this method would lead to many false triggers.

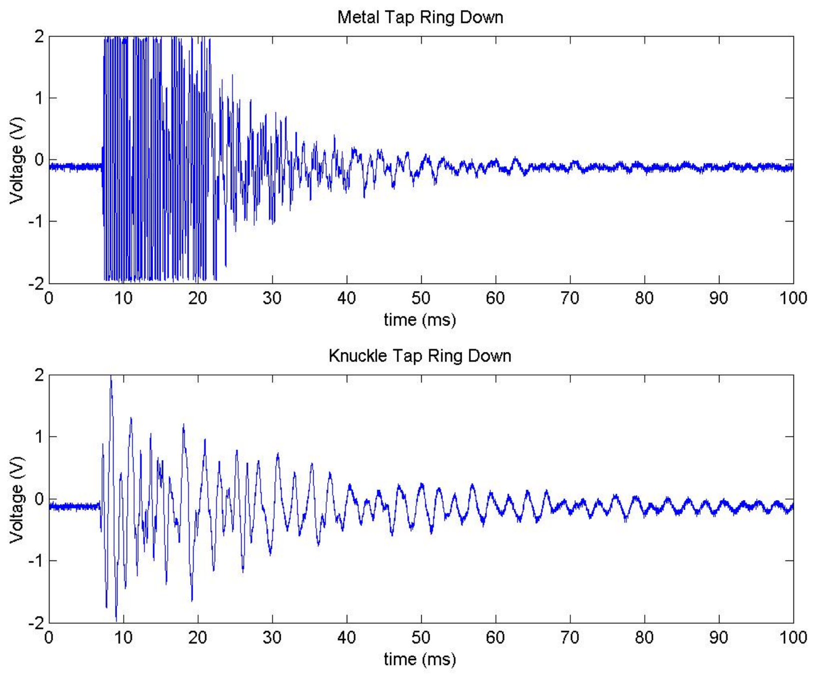

Figure 3-10: Ring down for a metal (top) and knuckle (bottom) tap

Figure 3-10 shows the "ring down" effect for a knuckle and metal tap placed at the center of a 143 cm $\times$ 143 cm window with sensors placed 35 cm from each corner. The window could ring for over 0.1 seconds like a chime (depending on the kind of glass and the mounting), causing false triggers immediately after the correct, initial tap was processed. Therefore, a dynamic trigger threshold was implemented, so that the knock threshold would immediately rise in magnitude after an event was triggered, and thereafter slowly return to the original base trigger threshold. Since any new events of significant intensity are well above the "leftover" signal from the previous impact, a trigger using the artificially higher threshold would mean a fresh knock has occurred. Once triggered, 400 samples (8ms) were stored, consisting of 150 pre-trigger samples and 250 post-trigger samples for each of the four contact microphones and the clap microphone. Since the low frequencies arrive much later, an extra 600 post-trigger samples were stored for the slow-response, low-frequency impact microphone (for a total of 1000 samples). Triggers were caused by exceeding the threshold on any of the contact pickups or the bash microphone signals. They were not used by the "clap" microphone, which only produces a veto.

3.3.3 Cellular Phone Noise

Because the Panasonic EFVRT series receivers and the preamplifier boards were sensitive to GSM cellular phone signals, the DSP was used to veto such signals. This was done by investigating the structure of the signals that which were picked up. Figure 3-11 shows the absolute-value waveform received for a common GSM signal.

The GSM pulse had a relatively sharp decay curve and was zero outside its pulse range. This was unlike a knock event, which took relatively longer periods of time to ring down. Thus, the DSP checks to see if the end of the sampled signal is at noise levels. This indicates that cellular phone signals, and not real knock events, are being picked up since the cellular phone signals decay much more rapidly.

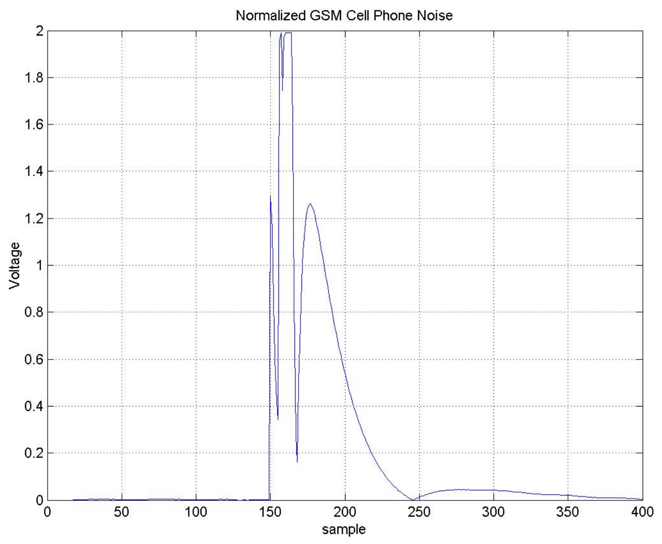

Figure 3-11: Absolute value GSM cellular phone noise picked up by the Panasonic EFVRT receiver and preamplifier (normalized)

3.3.4 Zero Crossing Count

A zero-crossing count was done before any waveform processing. This operation counted the number of significant crossings through the initial bias for the sampled waveform. A simple crossing count was done as follows:

(1) Calculate a threshold for noise $+d$ and $-e$ ($d, e \in R^+$), forming a transition band of width $d + e$.

(2) Increment the zero crossing count every time the signal alternated sides of the transition region.

This method only counted the dominant frequency in the system, since signals which had a frequency component smaller than the width of the transition region had little affect on the zero crossing count. It resulted in a fairly accurate tally of the number of zero crossings at the dominant frequency, thereby estimating whether the dominant frequency was low or high. This was used as a very simple means to calculate the dominant frequency of the flexural wave, so as to select the correct speed of the flexural wavefront.

3.3.5 Waveform Normalization

The initial bias was subtracted from the sampled waveforms, $x_a[n]$, before analysis was performed on them. This allowed all the sampled signals to be compared with each other without trouble. Because only differential time-of-flight analysis was implemented on a small frequency band, simple positive algebra could be used to avoid excessive use of absolute values on the DSP by finding the zero-referenced absolute value of the waveform. This is because the positive envelope for a given narrowband signal is approximately equal to its negative envelope.

The waveform was compensated by subtracting the initial bias in the signal (during the first fifty samples of the pre-trigger phase), and then the absolute value was taken to get the new waveform, $\breve{x}_a[n]$, as follows:

$$\breve{x}_a[n] = |x_a[n] - \frac{1}{50} \sum_{0<i\leq 50} x_a[i]| \tag{3.1}$$

In order to reduce noise in the system, the signal was conditioned with a simple first order FIR filter to acquire the normalized waveform, $\tilde{x}[n]$:

$$\tilde{x}_a[n] = \frac{\breve{x}_a[n-1] + \breve{x}_a[n]}{2} \tag{3.2}$$

3.3.6 Rising Edge Detection

When trying to determine which samples of each waveform originated from the source at the same time, the simplest solution would be to look for the spot where the signal rises above a threshold in all four waveforms. This is called rising edge detection, since it looks at the first significant rising edge of the waveform. However, it was found that rising edge detection was somewhat inaccurate, especially for knuckle knocks. A low threshold was susceptible to noise, while a high threshold was susceptible to missing rising edges due to attenuation of the signal. Knocks close to one sensor would have very different maximum magnitudes for the sensor that was close by compared to the sensor that was furthest away.

The method used here was to find the spot where the signal first surpassed $\frac{1}{4}$ of the maximum height of the sampled signal. From this point, the algorithm then backtracked n steps, before proceeding forward to find the first spot that rose above some lower threshold. In other words, in order to provide a more accurate rising edge estimate, a higher threshold was used first, preceded by a lower threshold crossing (in effect a double threshold level). The maximum n between thresholds was chosen to be 50 samples, an estimate for the length of two full cycles of the lower dominant frequency for the flexural wave.

None of these thresholds are related to the dynamic trigger threshold used in acquiring the samples and described in Section 3.3.2. However, since the lower threshold is a constant voltage (0.1V) above the maximum "noise" voltage in the pre-trigger phase, this method estimates the rising edge incorrectly less often than one cycle.

3.3.7 Heuristic Chi-Square Fit and Cross Correlation Method

While improving accuracy compared to the previous Responsive Window system for position determination, the rising edge method discussed in the previous section did not have good precision. One way to increase precision when using the rising edge method is to match the features (e.g. zero crossings and peaks of lobes) instead of just the rising edge in the normalized waveforms, $\tilde{x}_a[n]$.

It would be ideal to perform signal processing on the parts of the sampled signal that are least affected by dispersion or reflections. Due to dispersion, zero-crossings in those affected parts of the wavefronts are not reliable markers for time-of-difference calculations. Because dispersion causes higher frequencies to travel faster, the earliest part of the signal consisting of the highest dominant frequency was examined, resulting in the examination of a predominantly narrowband signal. In order to match lobes (and thus zero-crossings) for each of the signals, a sets of lobes most likely to have originated from the same part of the flexural wave were chosen from the sampled waveform . Hence, if $\tilde{x}[s_i]$ represented the sample that was the ith zero-crossing in the sampled waveform, the chosen lobes from $\tilde{x}[n]$ are those within the range $s_i \leq n \leq s_{i+N}$ such that, for some set of constants $\delta_1, \delta_2, \epsilon$, five heuristic conditions are met.

Heuristic #1

The first heuristic is based on the fact that it is not useful to compare noise with noise. Thus, a heuristic is needed that guarantees that the smallest lobe exceeds some minimum height:

$$\max_{s_i \leq n \leq s_{i+1}} \tilde{x}[n] > (1+\delta_1) \max_{0 \leq n \leq 50} \tilde{x}[n] + \epsilon \tag{3.3}$$

Equation 3.3 assures that the smallest lobe will be above some threshold determined by δ_1 and ϵ. For this implementation, $\delta_1 = 0.5$ and $\epsilon = 0.1(V)$ were used. The baseline threshold is determined from the first fifty samples of the pretrigger, before the wavefront arrived.

Heuristic #2

In order to insure that the lobes used had only one frequency component, only lobes that had no more than one point of inflection in them were used. Extra inflection points in the lobe would indicate the presence of a different frequency. However, saturated lobes (where the top of the lobe is flat) were included to accommodate intense impacts, as long as the lobe obeyed all the other heuristics.

Heuristic #3

Since it is obviously counter-intuitive to try to match lobes that are decaying in size with lobes that are increasing in size, it was arbitrarily declared that the sets of lobes that were selected must be increasing in size.

$$\max_{s_{i+j-1} \leq n \leq s_{i+j}} \tilde{x}[n] < \max_{s_{i+j} \leq n \leq s_{i+j+1}} \tilde{x}[n], 1 \leq j \leq N-1 \tag{3.4}$$

Equation 3.4 guarantees that each proceeding lobe has a higher amplitude than the previous lobe.

Heuristic #4

Because the bias was not always exactly centered between 0 and $5V$, the zero-referenced waveform had different heights for saturated signals. This was because the magnitudes of the negative lobes were generally different than the magnitudes of the positive lobes. We therefore used another heuristic to make sure that incorrectly passed Heuristic #3 were not picked.

$$\max_{s_{i+N-2} \leq n \leq s_{i+N-1}} \tilde{x}[n] < (1-\delta_2) \max_{s_{i+N-1} \leq n \leq s_{i+N}} \tilde{x}[n] \tag{3.5}$$

Equation 3.5 was required to make sure that inaccuracies created by normalization of the waveform did not affect the algorithm's performance on optimally choosing the best set of lobes. Setting $\delta_2 = 0.1$ was found to fix this problem.

Heuristic #5

In order to save on calculations, it was desirable to limit the number of lobes that were picked. Equation 3.6 was used to insure that at least one lobe was picked, and to keep calculations at a minimum (by constraining the number of lobes picked to be three or less).

$$1 \leq N \leq 3 \tag{3.6}$$

Having picked the lobes from each waveform, a method was devised to find which lobes most resembled each other. Assuming constant attenuation of the signal, the height of each compared set of sections was normalized. However, to figure out which lobe in the set of three lobes corresponded to a lobe in another set, the entire set of three lobes to the same height could not be normalized, since any comparison of three normalized lobes would likely lead to the conclusion that the first lobe in the first set correlated to the first lobe in the second set.

The solution to this problem is to compare sections of each set of lobes with each other. Each triplet of lobes was split into two pairs of lobes such that the two pairs of lobes consisted of lobes 1 and 2, and lobes 2 and 3 (assuming the triplet set of lobes had its lobes numbered in order: lobes 1, 2 and 3). Then, all combinations of pairs were matched up, and each match-up was normalized so that all pairs had a maximum height of one. This resulted in following mean-squared error function (chi-squared statistic) for each pair of lobe pairs for a range of values n:

$$\chi_{ab}[n] = \sum_{j=-(s_{i+N}-s_i)}^{s_{i+N}-s_i} (\hat{x}_a[j] - \hat{x}_b[j-n])^2 \tag{3.7}$$

where

$$\hat{x}_p[n] = \begin{cases} \frac{\tilde{x}_p[n]}{\max_{s_i \le n \le s_{i+N}} \tilde{x}_p[n]} & s_i \le n \le s_{i+N} \\ 0 & \text{elsewhere} \end{cases}$$

Next, the minimum of each set of pairs' mean-squared error function was found. The set of pairs that had the smallest minimum was selected as the set of two lobes that were most likely to match. A cross-correlation was performed on the two pairs of lobes, and the peak of the cross-correlation signal was returned as the differential time-of-flight. The cross-correlation, as conventionally defined in Oppenheim and Schafer, can be written as [7]:

$$\phi_{ab}[m] = \sum_{k=-\infty}^{\infty} \hat{x}_a[k]\hat{x}_b[m-k] \tag{3.8}$$

It should to be noted that all the calculations in this subsection need not be done to find the end result. In particular, it is not necessary to calculate every mean-squared error function for a large range of n in order to find its minimum. Instead, the Chi-square statistic can be calculated across a limited range of n where the minimum is most likely to occur. A sufficient sample range was found to be $n = x \pm 10$, where x is the average of the "center" of the two sets of lobes. Since $\hat{x}_a[k]$ and $\hat{x}_b[k]$ are finite signals, the cross-correlation can easily be calculated. The accuracy of this method is discussed in Chapter 4.

3.3.8 "Bash" and "Clap" Microphone Data Analysis

Because of the latency in the peak detector for the low frequency impact electrodynamic "bash" microphone and slower signal propagation, the "bash" signal does not arrive until about 500-600 samples into the post-trigger. Hence, it was necessary to store a 1000 length sample and find the maximum value, which was passed on as a parameter to the PC.

The crystal "clap" microphone data analysis was similar. It also passed the maximum value of the sampled signal to the PC.

3.4 PC Position Determination Code

The Position Determination Code, also written in C^{++}, was run on a PC attached to the DSP via a 115.2Kbit serial link. Upon receiving data from the DSP (indicating a knock event), it used the variables and parameters found by the calibration program, and the parameters from the DSP to calculate the knock position of the event.

The C^{++} code performed large-scale, memory-intensive calculations on the PC that would not be easily done on the DSP. This included the nearest neighbor algorithm, hyperbola intersection, and knock position determination. The other advantage to keeping this section of the code on the PC was that these calculations require input from the calibration program, which also ran on the PC.

3.4.1 Nearest Neighbors Algorithm

A five-nearest neighbors (5NN) algorithm, as described by Winston [6], was used to decide the difference between knuckle knocks and metal taps over a range of 6 variables: the number of zero-crossings detected by each sensor over a range of 400 samples and the time difference (in number of samples) of the arriving wavefront for each of two pairs of sensors. The first four variables provided a rough estimate of the frequency content at each of the four sensors; the last two variables were used to compare the frequency contents with the velocity of the bending wave, which is also dependent on the type of knock. The six variables were normalized into the same dimension, and the knock type was picked based on a majority of the five closest data points (in Euclidean 6-D space) in relation to the current knock's characteristics. Normalization was done so that no variable was weighted more than the other.

Setup

The variables z_1, z_2, z_3 and z_4 represented the number of zero crossings at sensors 1, 2, 3 and 4, respectively, variables c_1, and c_2 represented the time difference in number of samples between sensors 1 and 3, and sensors 2 and 4, respectively, $[0, m]$ represents the range of values for z_1, z_2, z_3, z_4, and $[0, n]$ represented the range of values for c_1 and c_2 be $[0, n]$.

Suppose they were 100 stored data points for metal taps and 100 stored data points for knuckle taps such that:

$$\mathbf{t[i]} = < z_{1:i}, z_{2:i}, z_{3:i}, z_{4:i}, c_{1:i}, c_{2:i} >$$

and for i= 1 to 100 these represent metal tap data points and for i= 101 to 200 these represent knuckle tap data points.

Given a new tap with characteristics:

$$\mathbf{t}[\mathbf{k}] = z_{1:k}, z_{2:k}, z_{3:k}, z_{4:k}, c_{1:k}, c_{2:k}$$

Then the five i are found such that the five distances $d[i]$, defined as:

$$d[i] = \sqrt{\sum_{j=1}^{4} \frac{(z_{j:k} - z_{j:i})^2}{m} + \sum_{j=1}^{2} \frac{(c_{j:k} - c_{j:i})^2}{n}} \tag{3.9}$$

are the five smallest distances over all i.

Hence, a simple majority is used - if there are more i selected below 100 than not, $\mathbf{t}[\mathbf{k}]$ is labeled as a knuckle tap. Otherwise, $\mathbf{t}[\mathbf{k}]$ is labeled as a metal tap.

In this way, the time difference (related to the velocity) is correlated to the zero crossings (related to frequency) to give an accurate estimate of the type of hit.

In practice, the number of calibration points does not have to be a total of 200 points, as is done in this setup. The number may go up or down with the size of the window, to avoid over and under-training the data. For this setup, 81 knuckle data points and 62 metal data points were used.

3.4.2 Calculating Hyperbolas

The main motivation here is to solve the problem of calculating a knock position for some given set of time differences between sensors, Δt_{13} and Δt_{24}. This was done by finding the intersection of the hyperbolas that represented the possible places of impact on an $x-y$ plane corresponding to the sheet of glass. The hyperbolas that are defined here consist of the set of points that produce a time difference Δt_{ab} between sensors located at (x_a, y_a) and (x_b, y_b) given an $x-y$ plane measured in centimeters. The point (0,0) is arbitrarily defined on the sheet of glass to be the lower left corner of the window, and one unit is defined to be the equivalent of one centimeter. Hence, to produce results matching these coordinates, the propagation velocity u, in cm/sec, of the dominant frequency of the flexural wave must be known. The difference in distance from the knock position (x, y) to the two sensors, a and b, will be equal to the distance the flexural wave can travel across the measured difference in time, which produces the following equation:

$$u\Delta t_{ab} = \sqrt{(x - x_a)^2 + (y - y_a)^2} - \sqrt{(x - x_b)^2 + (y - y_b)^2} \tag{3.10}$$

Because the DSP provides a time difference in samples, it is necessary to divide out by the sampling rate to calculate Δt in seconds.

3.4.3 Intersection of Hyperbolas

If the point of impact, (x, y) is unknown, it is only necessary to calculate the intersection of two hyperbolas in order to estimate (x, y), the position of the knock. This is done with an algorithm that minimizes the mean-squared error by selecting some (x, y) for the two equations:

$$E_{13}(x, y) = \sqrt{(x - x_1)^2 + (y - y_1)^2} - \sqrt{(x - x_3)^2 + (y - y_3)^2} - u\Delta t_{13} \tag{3.11}$$

$$E_{24}(x, y) = \sqrt{(x - x_2)^2 + (y - y_2)^2} - \sqrt{(x - x_4)^2 + (y - y_4)^2} - u\Delta t_{24} \tag{3.12}$$

that minimizes:

$$E(x, y) = E_{13}^2(x, y) + E_{24}^2(x, y) \tag{3.13}$$

When $E(x, y) = 0$, both E_{13} and E_{24} are zero, and thus the point (x, y) satisfies both hyperbolas. To solve for $E(x, y) = 0$ it is assumed that both hyperbolas are defined over some continuous range of x, for $m \leq y \leq n$. In this system, setting $m = -1000$ and $n = 1000$ was sufficient. Next, $E(x, y)$ was minimized over x for $y = y_0$ (where y_0 is the center y-coordinate of the glass), via a binary search. The nature of hyperbolas with foci on the diagonal corners of a rectangle require:

$$\frac{\delta E_i(x, y)}{\delta x} = 0$$

at a maximum of one point. $E(x, y)$ is minimized when x is set to x_0 (where x_0 is the value found previously when $E(x, y)$ was minimized over x for $y = y_0$). Because this recursively zones in on the hyperbolas' intersection, it is estimated that after about seven iterations on each axis roughly a cm-level precision will have been attained, since a binary search will have approximately halved the distance of the current guess from the actual intersection. However, if no intersection exists, the algorithm will diverge. This case can easily be checked by making sure the calculated intersection falls within the range of the window.

3.4.4 Knock Position Determination

Statistically, the chi-square cross-correlation method provides better precision (less than 2 cm away from actual position) 78% of the time, compared to the rising edge method, when used for calculating knuckle-based taps. However, it can sometimes shift discretely, if the wrong lobe is picked for correlation. To smear this effect, the resultant two estimates from both methods are averaged when their results are within a distance $\Delta d = 10$ cm from each other. When Δd is further than this threshold distance, it is assumed the wrong lobe was picked, and use the rising edge estimate alone. The result is an improved precision and accuracy over using only the rising edge estimate, or only the cross-correlation method estimate. Because of sampling rates, the chi-square cross-correlation method is ineffective for use with the metal taps. Therefore, the rising edge method is always used to calculate positioning for metal-based taps. Chapter 4 describes the results for these methods.

3.4.5 Low Frequency Impact and Clap Veto Microphone Data

A simple principle was used to figure out whether or not a triggered event needed to be treated as a low frequency impact or as an external "clap" veto. Because a low frequency impact would be picked up poorly by the four main piezoelectric sensors, compared to the signal on the electrodynamic microphone (similarly for the clap sensor for sharp external sounds), only the maximum recorded signal on the four main microphones was needed for comparison with the maximum for each of the two special microphones. If one of the special microphones had a higher maximum, then we would know that one of the special event cases had triggered the system. Care must be taken to adjust the analog gains for all the microphones so as to minimize the number of false triggers, since triggers can come from the four main contact microphones or the electrodynamic "bash" microphone. The analysis for this method is discussed in Chapter 4.

3.5 Window Calibration Code

The Window Calibration Program was written in C^{++}. It collected information to calculate variables for the PC Knock Position Determination program. These variables were stored in a file that was read by the PC Knock Position Determination software. The purpose of the calibration scheme was to gather the following information:

1. Location of the sensors

2. Data for nearest neighbors algorithm

3. Average velocity for different types of waveforms

4. Information for mapping points on the glass to pixels to be projected on the window

In order to gather information for mapping points on the glass to pixels for the projector, the calibration code necessitated the use of the projector. Therefore, the program was written on the PC.

3.5.1 Location of Sensors

The location of sensors played an important role in the ability to gather good data for the algorithms. It was found that sensors could not be arbitrarily placed. Therefore, code was written to take the size of the window as an input, and to give optimal points for sensor placement. Chapter 5 discusses how different sensor placements affect performance of our algorithms.

3.5.2 Determining Velocity at Relevant Frequencies

When an FFT of the waveform is taken, it is apparent that there are two major frequency components - one higher frequency for “metallic” taps and one lower frequency for duller, knuckle-based knocks. If the type of knock is known, the respective velocity u in Equations 3.11 and 3.12 is used. Therefore, calculation of u for each case is simple. Two points on the glass sheet are arbitrarily chosen that are relatively far from the center. Knowing these coordinates, and the coordinates of the sensors, twenty knock events are arbitrarily gathered for each position, and the time difference for each position is averaged. Since all the variables except u in Equations 3.11 and 3.12 are known, u can be solved for. The u's that are found for both positions are then averaged. Averaging the velocity estimates allows a better statistical accuracy to be obtained, since inhomgeneities in the glass will usually lead to slightly nonlinear average velocity.[8] Methods for compensating this phenomenon are discussed in Chapter 5.

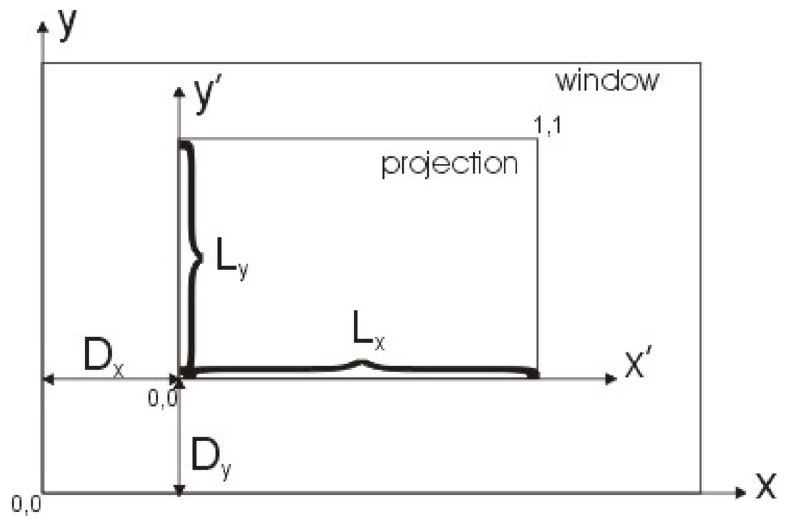

Figure 3-12: Translation from point on window to pixel on projection

3.5.3 Tap Types

For this section of the code, the calibrator was instructed to knock on the sheet of glass to provide data points for knuckle-based and metal-based taps to use with the nearest neighbors algorithm. This allowed the main program to figure out if the knock originated from the hand of a user, or from some foreign object that excited the higher frequency in the glass. Because different joints in the hand caused different frequencies, the calibrator was instructed to knock on the window using different knuckles, mainly the metacarpophalangael, proximal and distal interphalangeal joints (lower, middle, and upper knuckles).

3.5.4 Mapping from the Glass to Pixels

In order for the system to interact accurately with a projected image on the screen, it was necessary to translate the coordinates calculation for an impact on the glass into a specific pixel on the projector. This required a few variables that are easily attained by the calibrator - the dimensions of the projection onto the window, and the distance from the edges of the projection to the sides of the window, assuming a rectangular window. In the case where we are using a non-rectangular shaped window, the math can be generalized by superimposing a rectangular grid over it, and performing the calculations in the same fashion. A window with a projection area of $L_x \times L_y$ (in cm) and distances D_x, D_y from projection edges to the two window edges closest to $(x, y) = (0{,}0)$ would require a translation from window point (x, y) to the pixel point (x', y') (where x' and y' are normalized to one), would be related by the following two formulas:

$$x' = \frac{x - D_x}{L_x} \tag{3.14}$$

$$y' = \frac{y - D_y}{L_y} \tag{3.15}$$

Thus, points translated from within the geometric grid for projection would fall within a range of [0,1]. Figure 3-12 relates the relative variables described above. This calibration could be automatically accomplished by having the calibrator knock on a series of projected points.

Chapter 4

Responsive Window Performance

In this chapter, the performance of the Responsive Window system is analyzed. Because the structural-acoustic properties of every sheet of glass are different, the system's performance will change for different types of glass and different mounting arrangements. In this chapter, the performance of the system is examined across different media, venues, and mountings. The thinnest glass used, $\frac{1}{4}$ inch thick tempered glass was found to work the best, because the bending wave propagation speed is slower as the glass grows thinner. It was also found that thicker glass tended to have larger amounts of attenuation, thereby decreasing the sensitivity of the system. Mounting also changed the performance of the system. Mounting includes how the window is supported, as well as where and how the sensors are mounted on its surface. The positions of the sensors were critical in picking up clear signals, for which our algorithms would be able to pinpoint position correctly. Finally the effective sensitivity of the Responsive Window is discussed.

4.1 Tap Type Determination

In order to explore the performance of the Responsive Window's Tap Type Determination, the performance of the Nearest Neighbors algorithm is examined against the

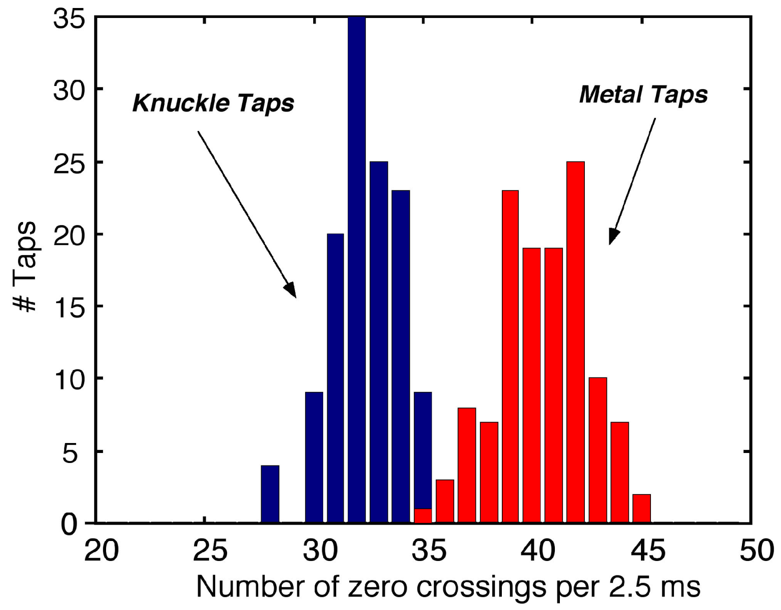

Figure 4-1: Histogram of knuckle and metal tap frequency

performance of previous methods.

Figure 4-1 shows a histogram for how well a simple summation of zero crossings was able to discern between knuckle and metal taps [9]. The graph shows an $88-99.5\%$ probability of correctly classifying taps.

When a new system was calibrated using the 5NN algorithm, the calibrator knocked on the sheet of glass using as many different knocking styles as possible to excite the glass in different ways. This enabled the gathering of data points required to distinguish between knock events that triggered higher and lower frequency flexural waves. The sampled waveforms for different types of knocks were investigated. Figure 3-8 shows the different types of taps that can be recognized by the system. The biggest concern was differentiating between a knock done by the distal interphalangeal joint (knuckle closest to the fingernail) and a metal tap. The only obvious difference is that the metal tap seems to have much more energy than the knuckle based tap. However, taking an DFT reveals that the knuckle tap actually

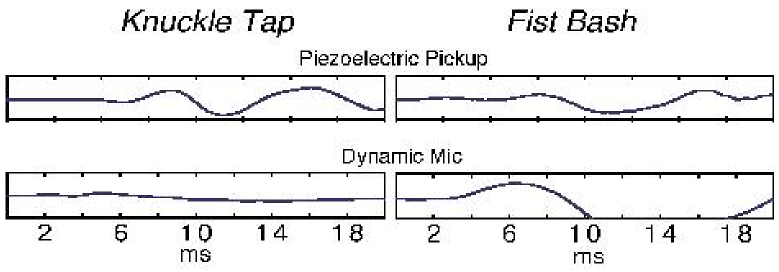

Figure 4-2: Piezo and dynamic responses to knuckle knock and bash

still has its dominant frequency at the same frequency as other knuckle-based knocks. Thus, the calculation should be done with a slower propagation velocity. By using the 5NN algorithm over the zero-crossing data and the differential time of flight numbers, it was possible to do a better job at discerning between these two events. Mainly, out of 1000 points tapped, only one point was incorrectly determined. This gives us a 99.9% accuracy.

Bash Determination

The bash microphone worked for the cases it was used for, and did not include a "bash" event when a regular knock occurred. Figure 4-2 shows that the electrodynamic microphone only responds to a fist bash, and not to a knuckle knock. [9]

4.2 Position Determination

When examining the performance of the Position Determination algorithm, it is instructive to first observe the granularity of the algorithms. Although the correlation analysis is capable of producing sub-sample interpolation, the rising edge procedure is constrained to finite granularity equal to that of the sampling rate. Therefore the source estimation can only be expected to be as good as the granularity. Figure 4-3 plots the positions that a given set of time differences (in units of samples) input to

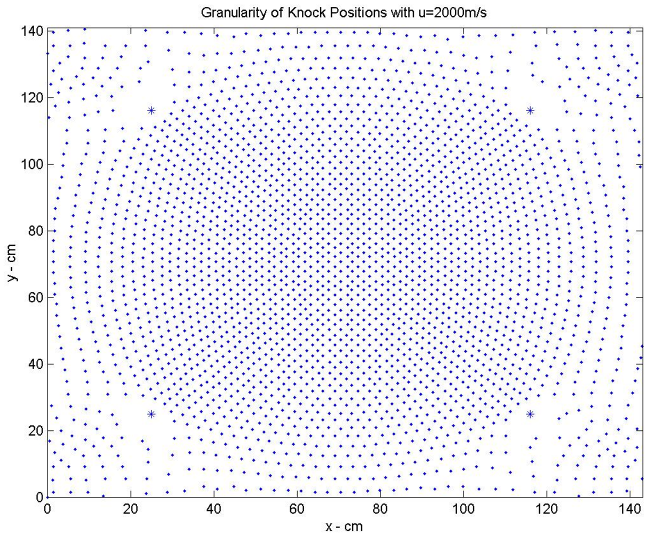

Figure 4-3: Granularity (for sample quantization) of Hyperbola Intersection algorithm for u=2000 m/s

our hyperbola intersection algorithm can output, assuming the flexural wave had a propagation velocity of 2000 m/s. The stars represent sensor locations. The granularity decreases further from the center of the sheet of glass. The granularity at the center for the setup in Figure 4-3 is about 2.5 cm. As the propagation velocity decreases, however, the granularity increases. Thus, for knuckle-based knocks, where the propagation velocity can range from 600 to 1200 m/s, the granularity will be much less; Figure 4-4 shows a zoomed in view of the center of the granularity plot for a propagation velocity of 600 m/s. Here, the algorithm sports better than 1 cm granularity.

Data from many different sources was looked at: two windows of different area and tempered glass, $\frac{1}{4}$ inch tempered glass, as well as one window with thicker shatterproof glass of 1cm thickness[9]. The plots show the reconstructed x, y positions for taps made repeatedly at the same positions, hence reflecting the intrinsic accuracy of the system. Results from the first window, shown in Figures 4-5 and 4-6, demonstrate

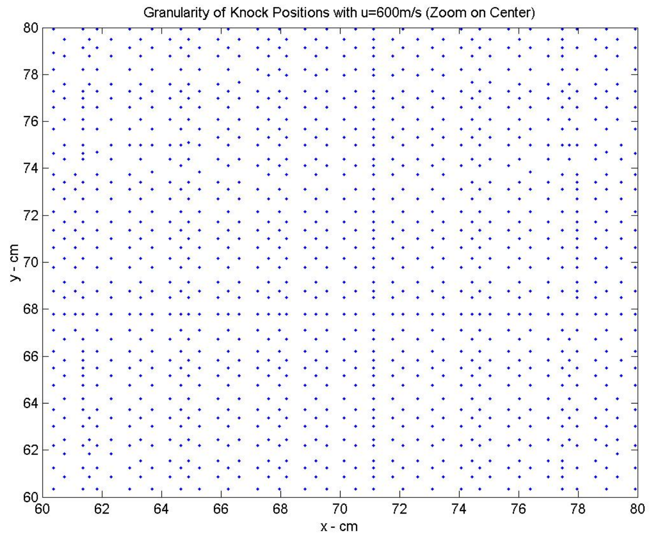

Figure 4-4: Zoom of granularity of Hyperbola Intersection algorithm for u=600 m/s

the resolution of the system on a 1 cm thick window. Across a microphone separation of 80 $cm\times$ 80 cm, a precision of $\sigma = 3.3$ cm was achieved for metal taps based on rising edge alone, and $\sigma = 3.5$ cm for knuckle taps based on the chi-square and cross-correlation method. Looking at the metal tap distribution plot, it is apparent that each knock position created an uniform distribution, while the chi-square cross-correlation method created tightly packed clusters that each had a very small standard deviation. This is because matching the wrong lobes would add or subtract the width of one lobe to the time differential, Δt_{ab}. However, the precision for the incorrect spot will still be the same. This suggests that if it were possible to match the correct lobes every single time, our algorithm would be extremely accurate.

Interestingly, the propagation wave speeds used to calculate the two types of knocks were quite high on this window. Metal taps produced flexural waves whose dominant component propagated at 1700 m/s, and knuckle-based knocks had a dominant component at 670 Hz. .

Figures 4-7 and 4-8 show knock distributions over a larger but thinner piece of

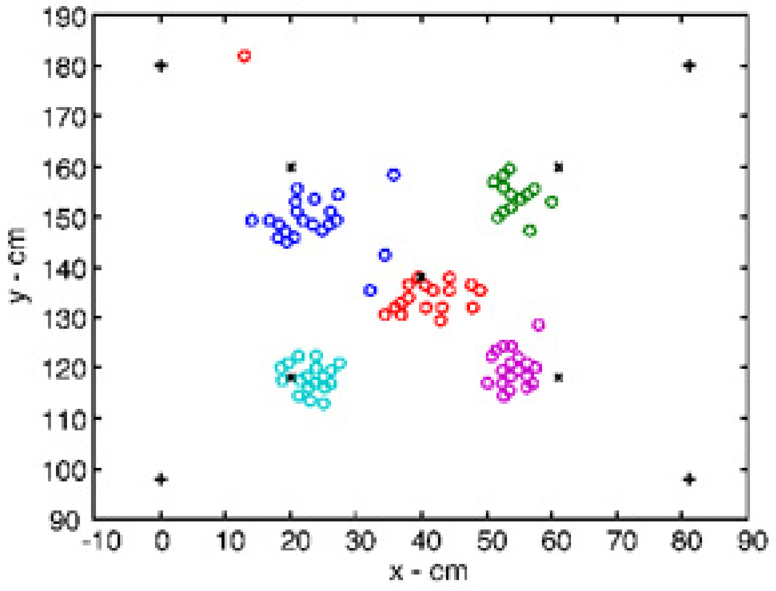

Figure 4-5: Estimated positions for metal taps at five positions for 1 cm glass

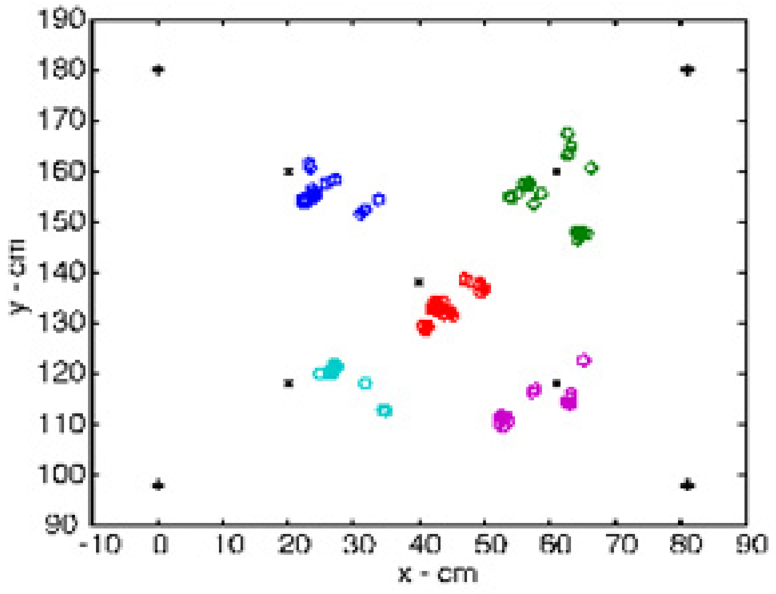

Figure 4-6: Estimated positions for knuckle taps at five positions for 1 cm glass

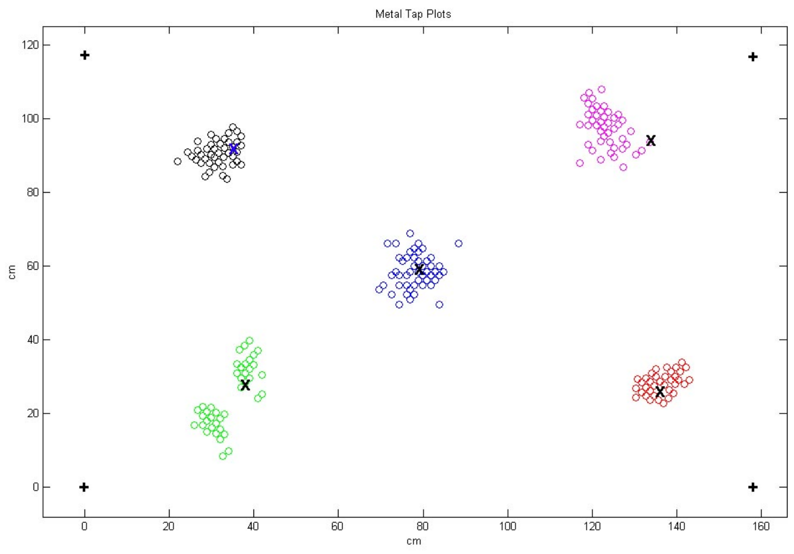

Figure 4-7: Estimated positions for metal taps at five positions for $\frac{1}{4}$ inch glass

glass. Because the speed of propagation for all types of impacts were slower through this glass, the overall sensitivity was increased, resulting in an average of $\sigma = 2.4$cm for both knuckle and metal events.

However, closer look at the data reveals that accuracy of the system was quite poor. Table 4.1 shows the statistics for this window. δ_{mean}, the accuracy, represents the distance from the mean position of all knocks on that point to the actual point. σ, the precision, is the standard deviation for the distances from the actual knock position. Appendix D explains the calculation of this variable. The last column represents how many knocks were properly processed, out of actual knocks physically counted. It is believed that this window's accuracy performed relatively poor because the sensors were placed too close to the corners. Furthermore, it was observed that the normalized waveform looked much clearer and well defined for the waveform picked up by a sensor located further away from the corner than those in the system with poor accuracy. This led to the conclusion that placing the sensors further from the corners

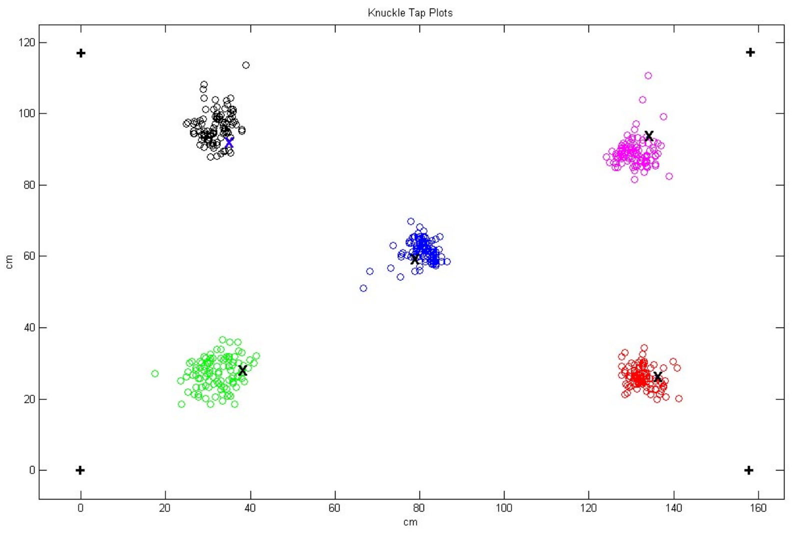

Figure 4-8: Estimated positions for knuckle taps at five positions for $\frac{1}{4}$ inch glass

of the window provides a better accuracy and precision for the position estimate. This is because the glass is more tightly pinned near the corner, attenuating and distorting the flexural waves traveling through that area.

Figures 4-9 and 4-10 show distributions for $\frac{1}{4}$ inch glass where the sensors are located 35 cm from the corners. The system was able to properly recognize to get all 100 taps for each point. Two points, however, were somehow gathered as extras. As the room was quite crowded at the time, it is expected that this was caused by a GSM cellular phone passing by, whose signal got mixed in with the knock, and caused strange properties to occur in the system.

It is encouraging to note that the accuracy for this system is much more accurate and precise at knock location. With an average accuracy of 2.0 cm and an average standard deviation of 1.6 cm, this system is accurate enough so that users should not be able to most errors, since the sizes of their hands are so large (larger than 4 cm radius).

Knock Type	Actual Position	$\delta_{mean}(cm)$	$\sigma(cm)$	Success/Knock
Knuckle	(38, 28)	5.9	2.4	102/102
Metal	(38, 28)	9.4	4.5	99/100
Knuckle	(35, 92)	5.4	2.6	99/99
Metal	(35, 92)	3.7	2.0	101/101
Knuckle	(79, 59)	3.1	2.4	103/103
Metal	(79, 59)	0.2	2.4	104/104
Knuckle	(136, 26)	3.3	2.1	103/103
Metal	(136, 26)	2.3	1.6	103/103
Knuckle	(134, 94)	5.8	2.8	104/104
Metal	(134, 94)	11.0	2.9	103/103

Table 4.1: Table of knock statistics for $\frac{1}{4}$ inch window with sensors closer to Center

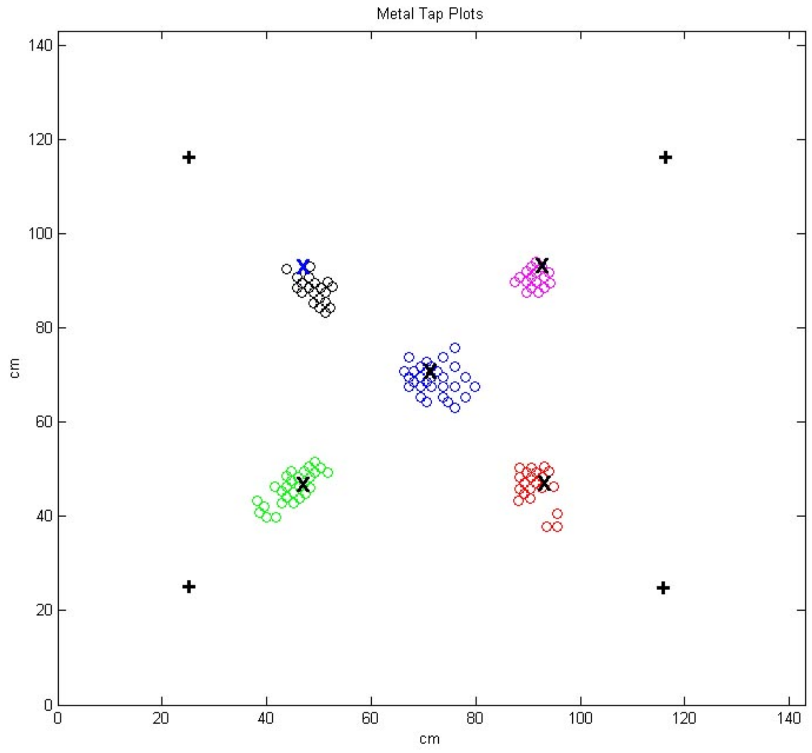

Figure 4-9: Estimated positions for metal taps at five positions for (sensors 35 cm from corner) $\frac{1}{4}$" glass

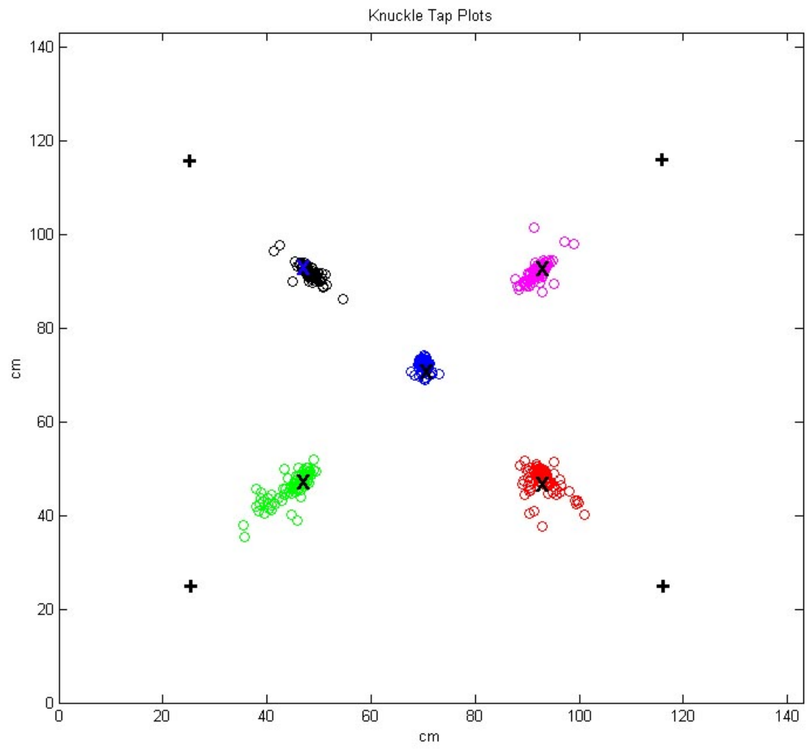

Figure 4-10: Estimated positions for knuckle taps at five positions for (sensors 35 cm from corner) $\frac{1}{4}$" glass

Knock Type	Actual Position	$\delta_{mean}(cm)$	$\sigma(cm)$	Success/Knock
Knuckle	(47, 47)	2.1	2.5	100/100
Metal	(47, 47)	1.6	1.8	100/100
Knuckle	(47, 93)	2.1	1.5	100/100
Metal	(47, 93)	4.5	1.3	100/100
Knuckle	(71, 71)	1.3	0.7	100/100
Metal	(71, 71)	1.8	1.8	101/100
Knuckle	(93, 47)	0.6	2.1	101/100
Metal	(93, 47)	2.3	1.6	100/100
Knuckle	(93, 93)	1.1	1.7	100/100
Metal	(93, 93)	3.2	0.7	100/100

Table 4.2: Table of knock statistics for $\frac{1}{4}$ inch window with sensors further from the corners

4.2.1 Tracking outside the sensor grid

On some windows, the sensors will be mounted so that there is still some window space outside the box created by the sensors. Figure 7-1 in Chapter 7 shows of one such window. Children were able to tap below the main sensor grid area. In this instance, it was only possible to accurately determine the x position of the knock. This problem is attributed to signal attenuation. Attenuation will cause the sensors farthest from the knock to lose virtually all of the incoming wavefront. Therefore, the only accurate data is that received from the two closest sensors. Because the intersection occurs closest to the two sensors that act as foci to the hyperbolas, the section of the hyperbolas defining the knock position will be influenced greatly by these foci. Thus, the sensors are able to accurately determine the x position, which is orthogonal to the direction between these two sensors.

4.2.2 Sources of Error

There are several sources of error which are likely to be responsible for inaccuracies in the system. The first source, was due to the physical limitations of system; because the DSP cannot sample any faster, it is impossible to obtain finer granularities for our hyperbola intersection algorithm.

The second source of error comes from the dispersive effects of the window. Even though a narrowband signal is assumed for the section that we perform calculations on, it is most definitely not narrowband. Even higher frequency components, which have very little gain, can corrupt the signal in a slight way. Without some method to compensate for the dispersion, it cannot be assume that the time-of-flight difference estimates are accurate. In some sense, the act of peak matching becomes meaningless if the transducers are at large differential distances from the knock; dispersion produces large changes in the nature and shapes of the peaks received at each transducer.

The third source of error comes from diffraction. Inhomogeneities in the glass, as well as inconsistent mounting of the window will cause flexural waves to spread

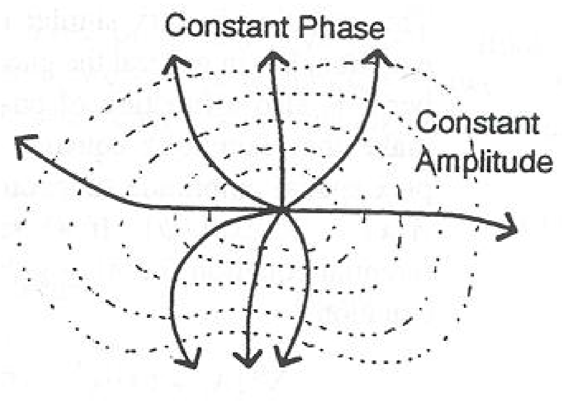

Figure 4-11: Rays of energy leaving a point source in an inhomogeneous medium

unpredictably in different directions [8]. The algorithm assumes that the flexural wave emanates from the source in an ideal fashion - in a circular fashion. Hence, constant velocity cannot be assumed if it is also assumed that the wavefront travels in a straight line from the source to the sensor. Figure 4-11 is taken from Johnson [8], and shows us how waves of energy can be distorted in nonhomogeneous media..

The last source is human error. It is not possible for the calibrator to hit the same point on the glass at the same spot each time. Compounded with the large surface area on a knuckle, ct extremely accurate results ($\sigma < 1\ cm$) for knock positions at one location.

4.3 System Latency

There were four respective contributors to system latency: time for flexural waves to travel to sensors, sampling period, data processing on the DSP, and data processing on the desktop. For a window of size 1 meter $\times$ 1 meter, the maximum distance a wavefront would travel to reach at least one sensor would be 0.707 m. If the propagating flexural wave was traveling at 600 m/s, the total time this would take is 1.2 ms. The sampling period had a post-trigger sampling time of 17 ms for the "bash" microphone and 5ms for the other five sensors. Using an oscilloscope to analyze latency between when a serial signal was sent and when the waveform was sampled, it was determined that the DSP took approximately 50 ms to process the tap. Finally, the computer (a Pentium III 1GHz machine) took about 10 ms to find the position. This adds up to a total of about 60ms of system latency. If the time it takes to render one or two frames onto the screen is added, at a 60 Hz refresh rate, it is expected that our system will take 80-90 ms to process the entire tap and produce updated graphics.

4.4 System Sensitivity

The system sensitivity is a factor that is difficult to measure in absolute terms. Because there is an adjustable gain stage in the analog signal conditioning electronics before the signals are sampled by the DSP, the system sensitivity levels can be set using both DSP code, and the gain levels provided by the analog electronics. However, it was observed that too much gain before sampling could cause early saturation for many hits (particularly due to the difference in sensitivity between metal and knuckle hits; this system must accommodate both). This can result in a very low dynamic range, since the input span for the DSP A/D converter was $\pm 2V$. Too much gain would cause the operational amplifier providing the gain to rail, and thus meaningful signal was lost. At the same time, too low of a threshold to trigger sampling and processing on the DSP caused "ring down" of signals to falsely retrigger the system to process the signal. Though most "ring down" false triggers were easily identified by looking to see if the rising edge occurred before the physical constraints dictated that they could occur, allowing the system falsely trigger this way can increase dead time. Figure 3-10 from Chapter 3 showed the "ring down" effect for a knuckle and metal tap. Although a metal tap saturates faster, it also has a shorter ring down time. This is because higher frequency components decay faster in glass, since their

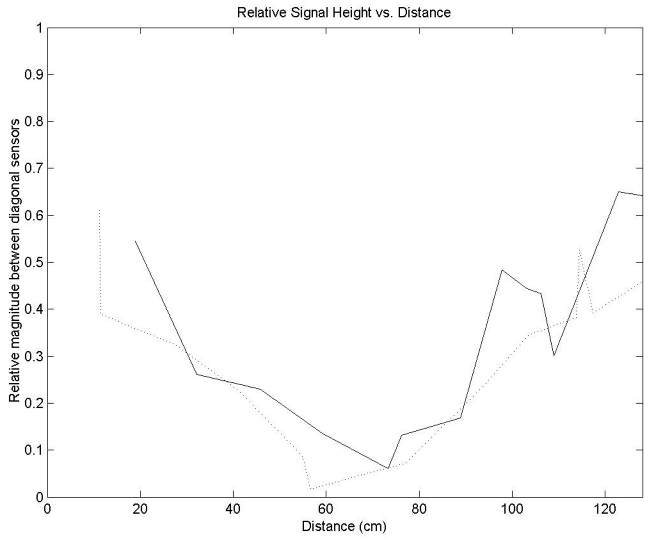

Figure 4-12: Relative signal height vs. distance

faster propagation velocity relative to lower frequency components cause them to be dissipated in the window mounting more quickly.

Since the knuckle tap takes a little more than 10 ms to reach noise levels, the dynamic trigger threshold allows the system to prevent itself from automatically re-triggering itself from the ring down. In this way, the sensitivity of the system was increased without having to increase the gain for any of the channels, which would cause signal loss for knocks with large amplitudes.

Figure 4-12 shows the correlation between relative maximum voltage levels picked up by two sensors versus distance from both sensors. The x-axis represents the distance the knock was from preset diagonal sensors. Two lines are plotted to represent voltage levels for the two diagonals of a rectangular window. The relative height, y was calculated as follows:

$$y = \frac{|\max \tilde{x}_a[n] - \max \tilde{x}_c[n]|}{\max \tilde{x}_a[n] + \max \tilde{x}_c[n]}$$

As one might expect, as the user knocks toward the center, the difference in maximum magnitudes decreases somewhat linearly. This could mean that the attenuation is occurring roughly linear for the given system. The distance between diagonal sensors is approximately 128 cm.

Chapter 5

Discussion

This chapter discusses the factors that were taken into consideration when redesigning the Responsive Window system. These included parameters that affected the acquired signals, such as the media type, dispersion and lossness, and sensor placement.

5.1 Dispersion and Attenuation

Most media in the real world are dispersive and lossy in nature. Lossy media causes signal components to lose strength over time. Thus, signal sources far from the sensor are difficult to discern from noise. Losses can vary with frequency, and in most mediums, higher frequencies are more heavily attenuated. Dispersive media causes wave propagation speeds to differ for different frequencies [8]. A common example of a dispersive object is a prism with incident light; it is visually apparent that the prism spreads the different wavelengths of light across a range of angles. Because dispersive media causes different frequencies to be phase shifted at the pickup point, it follows that each frequency component will also undergo different amounts of attenuation, dependent on the frequency [8]. Since some aspects of attenuation are related to dispersion, and dispersion is a major factor limiting timing accuracy, the implementation of several different methods to compensate for dispersion was investigated. Wave propagation can be modeled through a dispersive and lossy medium with a differential equation[8]:

$$\nabla^2 s = \frac{1}{c^2}\frac{\delta^2 s}{\delta t^2} + \gamma_1 \frac{\delta s}{\delta t} + \gamma_2 s \tag{5.1}$$

where ∇^2 is the Laplacian operator (defined in Appendix A), s is the displacement of the wave, c is the acoustic analog of the speed of propagation, and γ_1, γ_2 are constants. When $\gamma_1 = 0$, Equation 5.1 models a wave undergoing zero dispersion, and when $\gamma_2 = 0$, it models a wave with no attenuation.

5.1.1 Iterative Methods

A common method for dispersion compensation is the use of linear prediction to estimate the source position, given the current measurement and the previous measurement(s) for position of the source [8]. Compensation for dispersion occurs by iteratively recalculating the distance traveled by the source from its last known position. Most tracking models use state-space equations to recursively calculate the next position, and rely on the source moving predictably. Kalman filtering, Wiener filtering, and adaptive filtering are all models based on a linearization of the source movement [10].

Because knocks that are tracked are based on the whims of the user, it is not likely that the user's next knock can be estimated from a previous knock, unless the triggers are occurring at a steady pace. Comparing prediction errors based on these filters, it is unlikely that linear prediction in this fashion will do better than $\sigma = 2.5$ cm using the time domain based algorithms. Thus, an algorithm is proposed based on recursively estimating average velocity for selected frequency components for a single tap based on differential time-of-flights acquired by either the rising edge method or the chi-square cross-correlation method. By iteratively calculating the velocity for certain frequency components, any error in estimating the differential time-of-flight of each waveform will appear as a difference in velocity for the frequency components. A heuristic can be used to correct the time-of-flight difference between every pair of sensors and to iterate until the difference between the velocity vectors is minimized.

5.1.2 Dechirping Filters

Instead of doing calculations based on the time domain of the sampled signal, an iterative method could be used in the frequency domain to systematically phase shift different frequency components so that all the frequencies have the same phase, thereby creating one large rising edge in the time domain which could be used for timing analysis.

A very common area that uses dispersion compensation filtering is radar [11]. A pulse expansion is used to disperse a signal before transmitting. A pulse compression system is used to convert the received pulse into a single impulse for timing analysis. The "chirp" terminology comes from the meaning of pulse expansion. By dispersing a pulse into several components at different frequencies, several "chirps" are transmitted. Similarly, the pulse compression system is sometimes thought of as a "dechirping" filter. The pulse compression and expansion system is usually implemented in the form of a waveguide. The waveguide disperses the signal as it is transmitted; when received, the waveguide inversely compresses the signal.

Another common dispersion compensation method uses real physical objects with gratings or prism-like properties to slow faster frequencies down so that they are in phase. This physical method is commonly used in optical settings, which require a separation and slowdown of signals in the nanosecond or femtosecond range, before recompressing the signal back into its original wavelength [12]. There also exists types of sensors that use successive sheets of piezoelectric material to compress the signal [11]. The wavelengths that are compressed are on the same order of the spacing between sheets.

The analog of the pulse expansion and compression system for the Responsive Window would require a knock event to be modeled as an impulse, and to either create a sensor that performs pulse compression similar to the dispersive nature of the glass (pulse expansion). The sensor would thus require some sort of grating to achieve this result. One drawback to this method is that a different sensor would be required to complement each glass sheet, or at the very least, each glass sheet thickness, since dispersion is dependent on the thickness of the glass sheet. However, because there is no fixed range for detection, the sensor will not properly compensate for dispersion for every single knock. Therefore, extra computation would be necessary to compensate for dispersion in this setup.

The alternative to building a sensor with the appropriate properties is to completely perform the pulse compression on a computer. This would be more computationally intensive, but would allow the system to be more portable. The calibration program would have to be modified to calculate propagation speeds for different frequency components. One possible implementation is to use a matched filter [8] whose pulse compression is based on the dominant frequency(s) picked up by the contact sensors. An iterative search is done for the correct amounts of compensation for each major frequency component until the dispersion compensated signal matches a typical pulse, by passing the set threshold for the filter. To simplify the complexity of the model, the waveform is modeled as the sum of only its dominant frequencies and the other lower magnitude frequency components are regarded as colored noise. Therefore, the sampled data needs to pass through a whitening filter before being passed through the matched filter [8].

5.2 Simpler Timing Analysis

A simple idea that would apply both the time-of-flight calculation methods described in Chapter 3 is to use the zero crossing nearest to the rising edge. This is easily done by taking the rising edge and backtracking to the nearest zero-crossing. The only caveat here is that if the rising edge occurs in a region that is the sum of several dominant frequencies, the zero crossing is inaccurate. Therefore, it would be best to use a bandpass filter to narrow the signal's bandwidth, and to then trace from the corresponding rising edges in time in the full signal backwards to the nearest zero-crossing in the narrowband signal.

5.3 Window Sensor Placement

It was desirable to minimize the amount of sensor clutter on the window, while optimizing the sensor location, so four sensors were used to calculate position of impact on the window. Multiple sensor placement positions that were considered:

(1) One sensor in each corner

(2) One sensor at the center of each side

(3) A pair of adjacent sensors at two locations

Figure 5-1 shows the different window sensor placements that were considered.

The advantage of placement (1) is that the sensors are least conspicuous in this setup. However, since the sensors are in the far corners, they may not always pick up desired clean signals. The mounting of the window could cause the sensor to be stiffly "pinned" by the two adjacent sides of the window. This pinning could cause the sensor position to result in a reduced sensitivity for some frequencies coming from specific impact sources on the window. It was observed that higher frequencies were often attenuated to the point where the corresponding signal components looked like noise. Thus, a solution is to place the sensors further away from the corners, so that they will not be "pinned" by two orthogonal sides of the window.

The primary advantage of placement (2) is that the sensors are now only pinned to one axis - the side that the sensor is placed on - allowing for a more predictable frequency response. The other advantage of this setup might be that the sensitivity

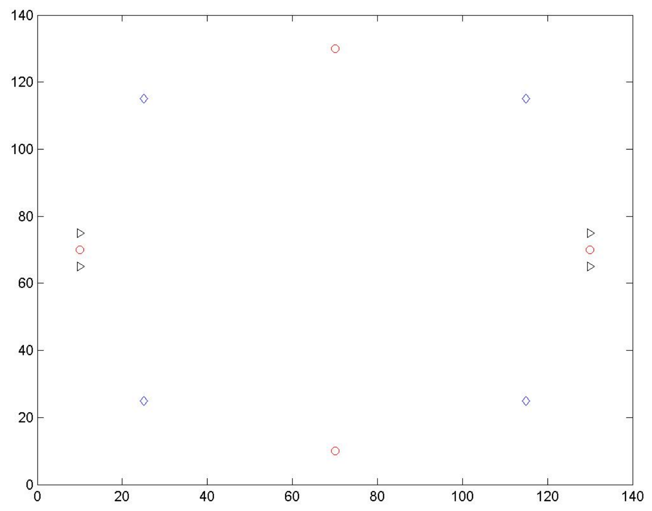

Figure 5-1: Different window sensor placements: placement 1 - blue; placement 2 - red; placement 3 - black

of the system is greater than placement (1), since all four sensors are now placed more closely to the center, and the four sensors are not placed on top of nodal lines for principle modal frequencies in the glass. However, a simple rotation of the sensor placement will reveal that placing the sensors closer to the center in the formation of placement (1) will result in the same sensitivity, in the tracking area sense (based on constant attenuation). Another big disadvantage to this placement scheme is that it forces a smaller area for video projection, especially to avoid projecting over the sensors. Empirical data also showed that the accuracy and precision for the Responsive Window using placement (2) was much worse than placement (1).

Placement (3) allows us to track using phase calculations. Since each pair of sensors are very close to each other, the waveform picked up by each sensor (in a given pair) should be very similar. Hence, a cross-correlation of the signal should easily reveal the phase difference, and two different time-of-flights should easily be able to calculate a point of impact. However, for large distances, Δt becomes very small, and therefore low sampling rates will not have the resolution required to resolve between points that are relatively far away. This problem can only be solved by sampling at a higher frequency.

5.3.1 Sensor Proximity to Window Edge

Empirically, the distance of a sensor to the window edge is highly correlated to the ability of receiving clean signals. The further the sensor is from the edge, the less likely reflections from the window's border are able to double back and get picked up by the sensor. This is because the reflection will be weaker (some of the energy in the wave is dissipated in the mounting) than the incoming wavefront, and attenuation will cause the signal to diminish rapidly from the edge of the window. For a window with $\frac{1}{4}$ inch thick glass, it was found that placement (1) will work well if the sensor is at least 20-25 centimeters away from the two adjacent sides of the window. We estimate that for a thicker pane of glass, since the average propagation velocity of all frequency components is higher, the distance from the edges may correspondingly need to increase, although changes in attenuation for thicker glass, and the manner of mounting may help here.

5.3.2 Valid Window Tracking Area

It was interesting to study how large of a tracking area each placement scheme would be able to cover. To begin, it was assumed that a signal traveling a distance of x would be attenuated to the point that the data was unusable. From this assumption, it was realized that the maximum amount of tracking area occurs when the sensors are placed in the formation of a square, if it is required that all four sensors require a valid signal. Thus, there is no obvious advantage between placement (1) and placement (2) for this heuristic. Placement (3) is at the worst disadvantage here, since it does not even form a square to maximize tracking area.

5.3.3 Nodal Behavior Effects

On square sheets of glass, there are two principle nodal lines, along the diagonals connecting both pairs of corners. Nodes represent “dull” spots where the modal behavior of the glass causes excitations on the glass to have little or no behavior for the corresponding modal frequency. (Consider observing how sand settles into a fixed harmonic pattern atop a vibrating plate.)

However, not all of these natural modal frequencies are dampened at those principle nodal lines. Cremer [13] explains that knocking on different points will excite different modal frequencies. It is hypothesized that the modal frequencies that are excited are the ones closest to the spectral content in the knock itself.

A further examination of the DTFT for sensor data from sensors placed on or off the nodes in the glass reveals that the frequency response is different. Since placement (1) places the sensor on the principle diagonal node, it is expected it to have a different frequency response than a sensor not on the diagonal node. Indeed, Figure 5-2 shows the dominant frequency in the sensor for placement (2) to be somewhat dampened in the frequency response for placement (1).

Thus, by placing the sensors on the nodes on the diagonals of the glass, some lower frequency modal vibrations in the system will be eliminated, resulting in a larger magnitude response in the next largest modal frequency. Because the frequency response of the receivers are limited at the lower range to 300 Hz, it seems that it would be better to do time domain analysis on a higher frequency signal. Mainly, placement (1) is believed to be more effective than (2) and (3), since the sensors in placement (1) are placed so that the lower modal frequencies will be naturally filtered out.

The chi-square cross-correlation algorithm is less suited to process a signal dominant in the lower frequencies around 400 Hz, because knocks exciting low frequencies will result in a slowly oscillating signal. It is difficult to define lobes for the algorithm in such a slowly moving signal; at 400 Hz there are an average of only two full

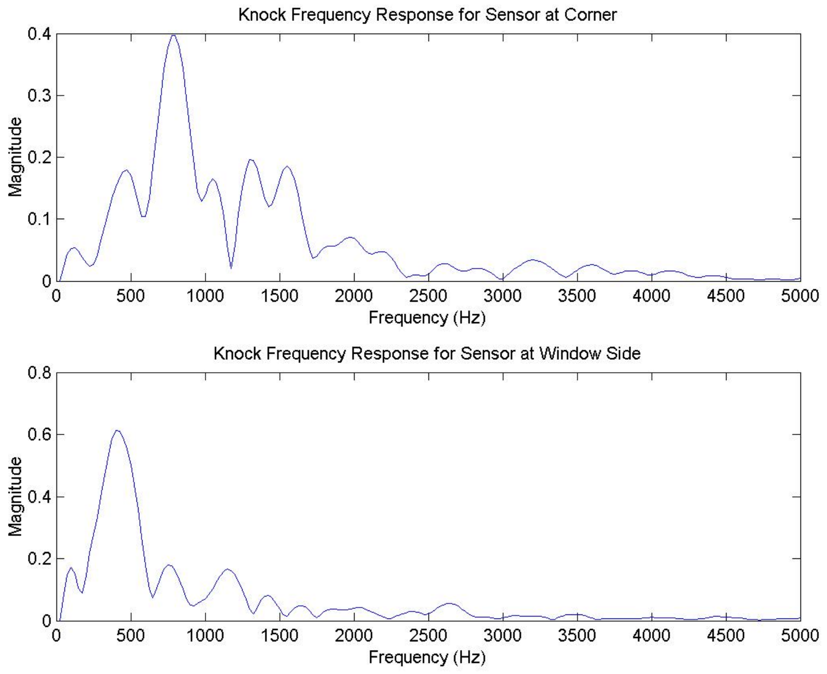

Figure 5-2: Comparison of frequency components for different sensor placement. placement (1) (top) versus placement (2) (bottom)

cycles of the waveform for the 250 samples representing the post-trigger phase. The rising edge algorithm will also become less accurate for lower frequency signals, since the rising edge becomes more ambiguous for slowly oscillating waveforms which have undergone attenuation.

5.3.4 Choosing the Best Placement

It would seem then, that placement (3) is far superior to placement (2), because the cross-correlation of two relatively similar signals will define a nice peak that will define the phase difference. However, due to the nonhomogeneous structure of glass (e.g. regions where the glass has been under stress), it cannot be assumed that the wavefronts will be shifted versions of each other for placement (3) for each pair of adjacent sensors. If the difference between the magnitudes for each sensor is modeled as having added noise proportional to the distance to the sensors from the position where the knock occurs, it is clear that a bigger distance between the knock and the sensors will translate to more difficulty in obtaining an accurate phase difference from a cross-correlation at each pair of sensors. This is because for large distances from the array's broadside, the distance a wave must travel to each sensor becomes roughly equal. In short, for large distances, placement (3) will return time differences close to zero for knocks occurring far away from the sensors. Phase measurements for measuring θ, the offset angle of the knock from the sensor array will also be a big problem for large distances r, since differential times will be minimal. The only way to solve this problem is by increasing the sampling rate of the system, thereby creating a finer timescale to differentiate the phase.

If the characteristics of the sampled waveform are examined for a sensor that is very close to the source, it is apparent that the system quickly rises to saturation. This is because the distance to the sensor is so small that the dispersive effects of the glass are minimal. Thus, it is important to balance the distance from the sensors in order to get relatively similar data versus the need for sensors to be close enough to the sensor so that our system has good sensitivity. For these reasons, for most window sizes and the algorithms defined in this thesis, placement (1) has be selected.

5.4 Sensor Characteristics

The types of sensors used to pick up data for waveform analysis are almost as important as the algorithms themselves. The ideal sensor should be easily adhered (with super-glue) to the glass surface, so that most of the energy in the flexural waves can be transferred and picked up by the sensor. It was found that tightly pressing the sensor against the glass using tape is not as effective in getting a waveform that is clean. Also, super-gluing the sensor to the media gives relatively stronger signals than those picked up by a sensor tightly pressed with tape. Therefore, the SNR for superglued sensors is also significantly higher.

The material type of sensor casing should also be considered. Sensors with a metallic shielding, such as those of the earlier Panasonic EFVRT series, were reusable, because the it was relatively easy to remove the superglued sensor from the glass without bending or destroying the microphone. The newer EFVRT series, however, while lighter, were also more fragile. These sensors seem to be one-time use for the system setup, because removal of the sensor from the glass usually caused the sensor's destruction of the sensor.

The frequency response of a sensor was also very important. A poor match in response between knuckle knocks and metal taps can lead to insensitivity.

Mounting the Sensors

The way sensors were mounted to the glass seemed to dramatically affect data pickup. Tightly pressed receivers did not seem to couple to the glass as well as superglued sensors. However, there seems to be a limit to how much glue can be applied to the system. Different amounts of applied glue seem to affect the frequency response of each sensor, perhaps because of damage to the sensor elements as glue leaches into the device or changes in spectral sensitivity as air holes on the sensor are obscured.

5.5 Glass Thickness

Because there are many types of glass, the amount of dispersion in each piece of glass can vary. Mainly, as glass thickness increases, flexural frequencies tend to travel faster through the media. Thus, in order to avoid having the propagation speed of the highest frequency being too fast to sample properly to get an accurate position estimation, the glass thickness would have to be kept to a tolerable limit. The observations made in Chapter 4 seem to point to the fact that tempered $\frac{1}{4}$ inch thick glass is the best media for the Responsive Window; it is thin enough to produce good results, but not so thin as to easily break.

The other issue with glass thickness involves the amount of discomfort the user can tolerate. A thicker sheet of glass translates to a relatively "harder" piece of glass, since the amount of mass that has to excited increases proportionally. This means that users may be less willing to interact with the glass because it hurts more to knock on it. Fortunately, it seems that the Responsive Window is most well suited with thinner sheets of glass, though it does accommodate comfortable knocks on all pieces of glass that were used by increasing the sensitivity accordingly for thicker windows. Of course, it does not work on double-paned windows, because of lack of contact between the two sides of the window.

Chapter 6

Radar Rangefinders

Radar was selected over other methods for rangefinding for several reasons in this application. First and foremost, radar is not lighting-dependent, as cameras are. The lighting situation for a system placed behind a highly reflective piece of glass will mostly likely be poor and dynamic. Hence, lighting-dependent methods may not be as effective as radars when lighting conditions became poor. Radar also reflects off skin, and is not affected by clothing and varying color. Radar can also see through projector walls and non-conductive media.

There were four radar systems investigated for integration with the Responsive Window - two Tank Radars made by Saab Marine [14], sponsors at the MIT Media Laboratory, which were designed to measure liquid levels in large fuel tanks (e.g. at an airport), Micro-Impulse Radar developed by Lawrence Livermore National Laboratory, and a Doppler Radar system developed by the Responsive Environments Group at the MIT Media Laboratory. The factors that went into picking the radar system were portability, availability, and complexity.

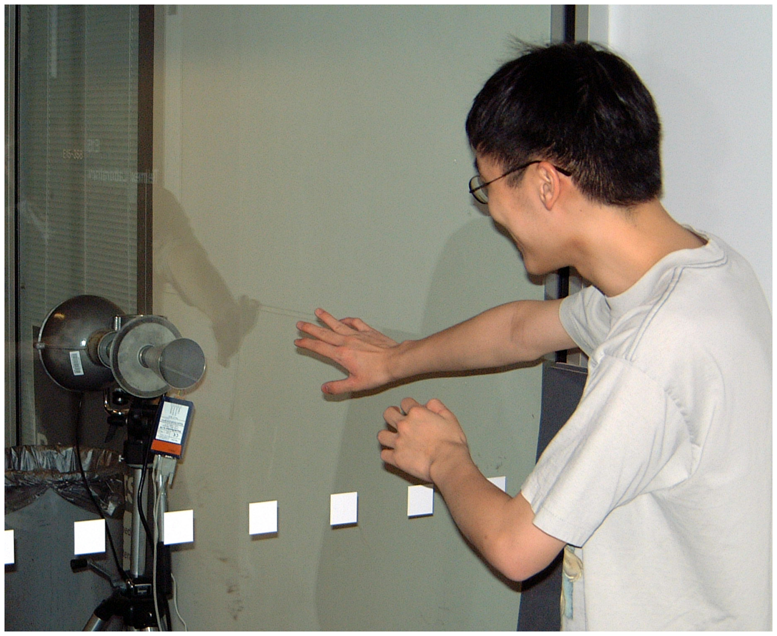

Figure 6-1: Using the Saab Tank Radar Pro as a radar rangefinder

6.1 Narrowband Tank Radars

Two narrowband radars made by Saab Marine that were used to detect liquid levels in a tank were investigated. These seemed well suited for this application, as the amount of energy output by the system was well within safety limits, and the range of operation was well matched to room-scale sensors.

The first system provided by Saab used a swept Doppler technique [14] to discern distance. However, the waveguide packaged with the device was massive, and the device was very unwieldy. Because the system had many more features than were required (it was a commercial product dedicated to its application), the refresh rate of the device was unsatisfactory (it had about a 2 Hz refresh rate and system latency of about one second), and thus was difficult to adapt to use with the Responsive Window. Figure 6-1 shows a picture of a theoretical use of the unwieldy radar system being tested by a cooperative user.

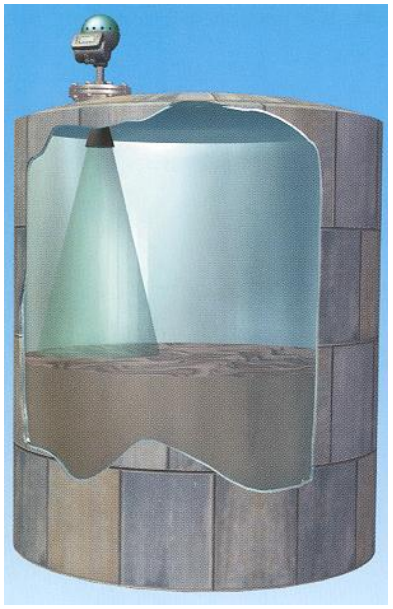

Figure 6-2: The commercial use for the Saab Radar Tank Pro

This tank radar had mixed results. Accuracy was somewhat limited, and its system latency was somewhat high. The extreme near field visibility of the radar system was mixed, with the optimum measurement distance for the system at 4-10 meters. The radar did not provide very stable results, even at this range. The measurements would jump erratically for people walking within its tracking area, most likely because of the slower refresh rate of the system. The other factor contributing to error is that the radar was designed for use within a tank of specified shape, and homogeneous wall structure. The beamwidth was also quite narrow. It was easy to "trick" the radar system by walking in and out of the radar's beamwidth. Though it was possible to get this system working in early tests, difficulty in adapting it to this application encouraged the testing of other devices.

A second improved tank radar was promised, but not delivered in time for thorough testing. This tank radar was similar to the original tank radar, but was smaller, and had a better interface [14]. Operating at a frequency of 10 GHz, the Saab Tank

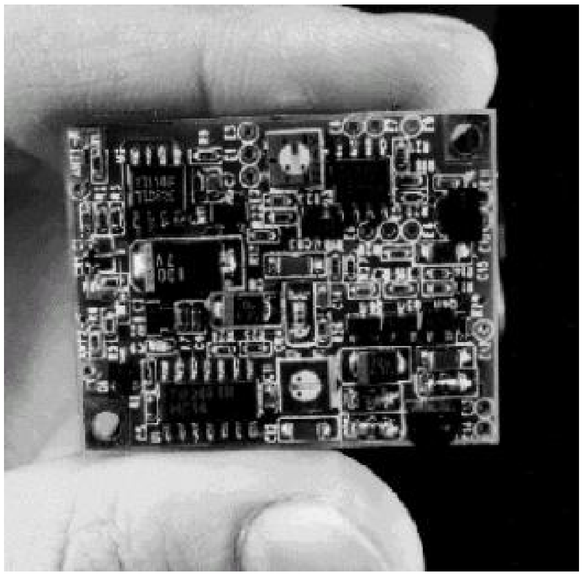

Figure 6-3: Micro-Impulse radar

Radar Pro used several different types of algorithms to improve accuracy. It combined the FFT, Frequency Modulated Continuous Wave (FMCW) Method and an echo compensator to calculate the liquid level in the tank. A serial port interface allowed several Tank Radars to be operated at the same time, if needed. The company boasted a 5 mm accuracy range. Figure 6-2 illustrates the radar in a tank liquid level detection setting. Further evaluation of this radar may prove beneficial.

6.2 Micro-Impulse Radar

The second radar system was one based on the technology promoted by Lawrence Livermore National Laboratories. The device, called *Micro-Impulse Radar* (MIR Technology)[15], used simple off the shelf components to create a radar system that compared randomly output impulses in a broad bandwidth with the signals received. This would give an accurate estimate of distance through simple time-of-flight calculations, and would tend to be immune to interference that could be caused by a

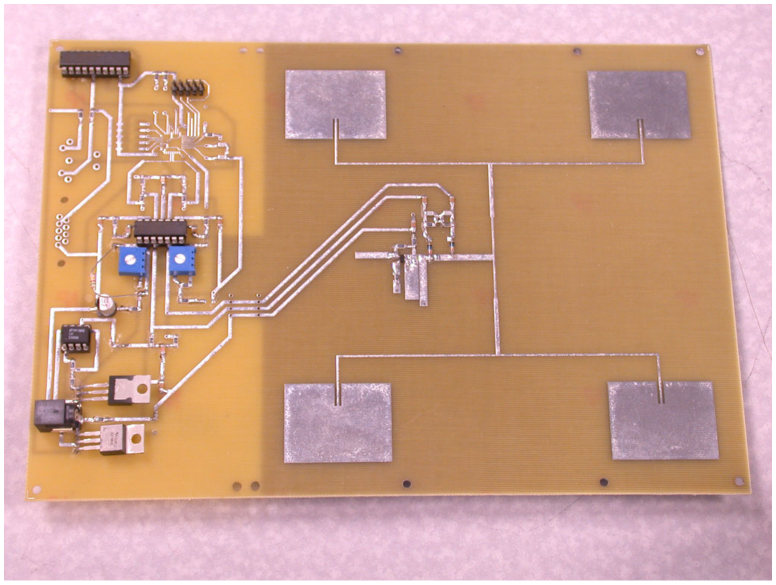

Figure 6-4: Doppler radar developed at the MIT Media Lab

narrowband radar such as the tank radar made by Saab. Figure 6-3 demonstrates the relative size of the Micro-Impulse Radar Unit. One major advantage of the system was that it was an ultra-wideband device, so that noise from other devices operating in some frequency range did not jam the radar. Although an early prototype of such a system has been built [16], the design of this system and its antennas is not yet ready for this application.

6.3 Doppler Radar

A much simpler radar was designed for this application, though this radar system was not a range-finding system. Developed at the MIT Media Lab, the Doppler radar was able to distinguish motion within the immediate surroundings by making use of the Doppler principle [17]. Figure 6-4 shows a picture of the prototype for the Doppler radar. One major drawback of the system was that the system was only able to distinguish the motion, but not the position of the user within the surrounding

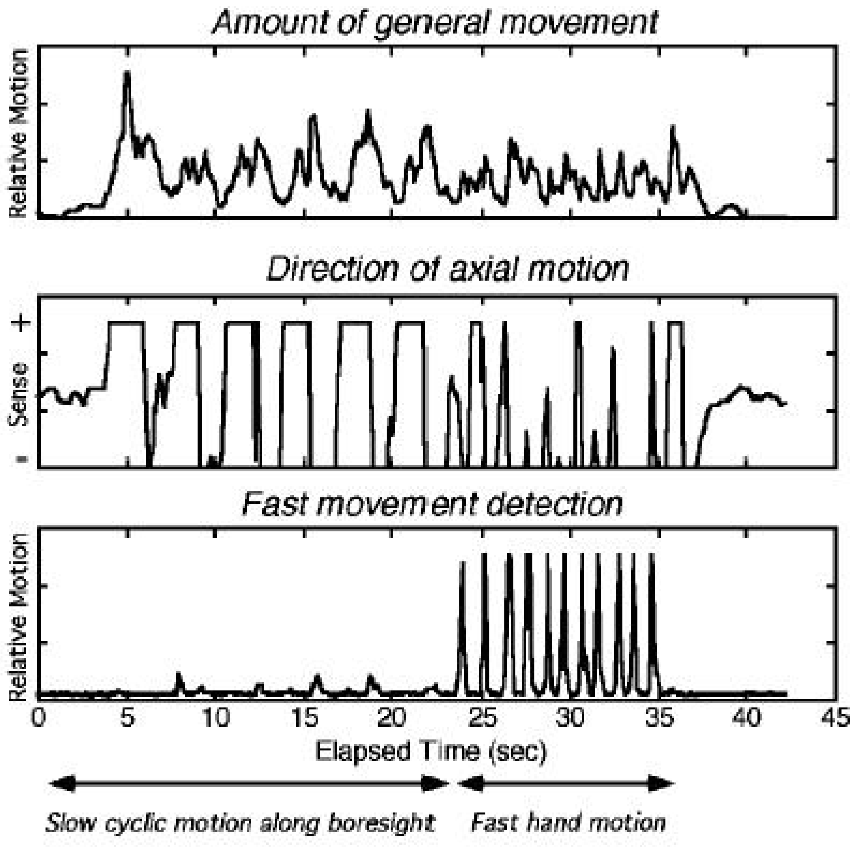

Figure 6-5: Doppler radar measurements

environment. Figure 6-5 demonstrates the data recorded by one Doppler radar.

Work is now being done to modify this system so that it can also detect position. The main difficulty is rooted in the fact that the system detects distance through attenuation of the signal; highly reflective objects (metallic objects) are perceived as much closer than similarly distanced objects that absorb the signal (such as concrete).

One possible method to resolve position is to use multiple Doppler radar units (> 2) to track user movement, and to estimate position through examining multiple velocity vector measurements. Though this will not solve the problem of stationary targets, it should resolve the position problem when the user makes gestures, since different measurements of changing velocity from two different directions can be mixed to reveal position of moving objects. A system is being developed by NASA to measure wind speed and wind vectors by using two Doppler Wind LIDAR (laser pulse guided systems) to sense velocity change [18].

Due to availability, portability, and ease of use in the design, the Doppler Radar system was chosen for future integration with the Responsive Window.

Chapter 7

Applications

The applications of the Responsive Window System detailed in this thesis include the use of the window as an Interactive Drawing Exhibit, a Store Front Display, and the more general Knock Driven Display. When used in conjunction with a radar-rangefinding device, the entire setup will be able to drive a more “sentient” system. Installations with the Responsive Window System have already been presented at three locations, including Boston, New York, and Linz, Austria. Each exhibit is discussed, along with how users responded to such a system. Finally, the new system, which will be debuted at the 2002 SIGGRAPH Convention Emerging Technologies Exhibition in July, is discussed.

7.1 Interactive Drawing Exhibit

The Interactive Drawing Exhibit for the Responsive Window [19] was debuted in Linz, Austria at the Ars Electronica Museum on August 31, 2001. It is a semipermanent installation that debuted in the 2001 Ars Electronica Festival as the “Responsive Window” [19]. The glass used was 1 cm thick, and was supported by two large columns on its left and right. It was free of constraints at its top and bottom. A device called a “holoscreen” was mounted behind the front of the glass, and a

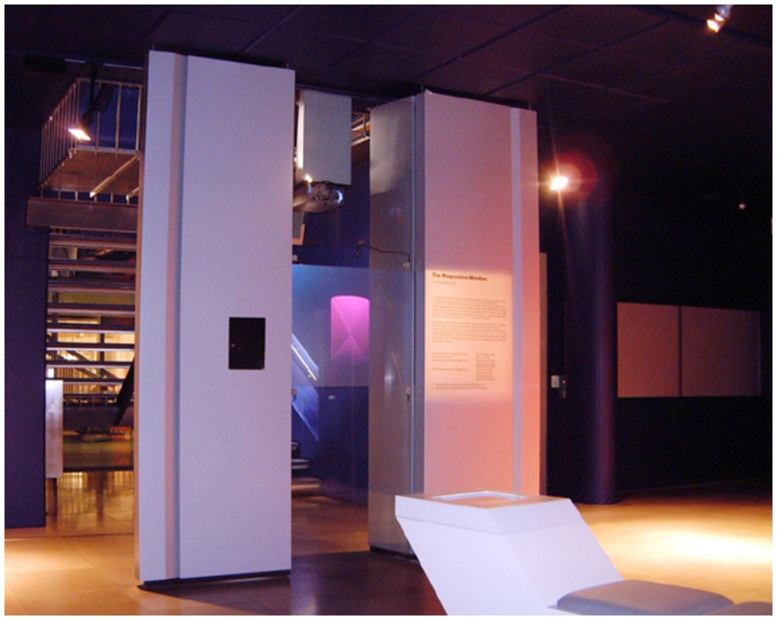

Figure 7-1: Interactive Drawing Exhibit at the Ars Electronica Museum

projector hanging from the ceiling cast graphics onto the holoscreen. Because of the nature of the holoscreen, users were both able to see through the projection and see the projection itself. The clap microphone was placed on the very top of the right pillar and the low frequency "bash" microphone was placed on the center of the glass, just above the top of the holoscreen. Figure 7-1 shows the setup from a distance - it is a piece of 1 cm thick glass, 1 meter in width, and 3 meters high upon which the sensors were affixed as mentioned previously. Figure 7-2 shows a close-up view.

The graphics that we were displayed resulted from a collaboration with Ben Fry from the MIT Media Lab's Aesthetics and Computation Group. It was essentially a simple knock-driven drawing program that illustrated the full responsiveness of the system. One knock generated a circle centered at the impact coordinate that oscillated as a function of knock intensity. Second and third knocks extruded a rotating 3D object, with dimensions determined by the knocks' coordinates. The rotation rate and fill intensity were similarly a function of the intensity of the knock.

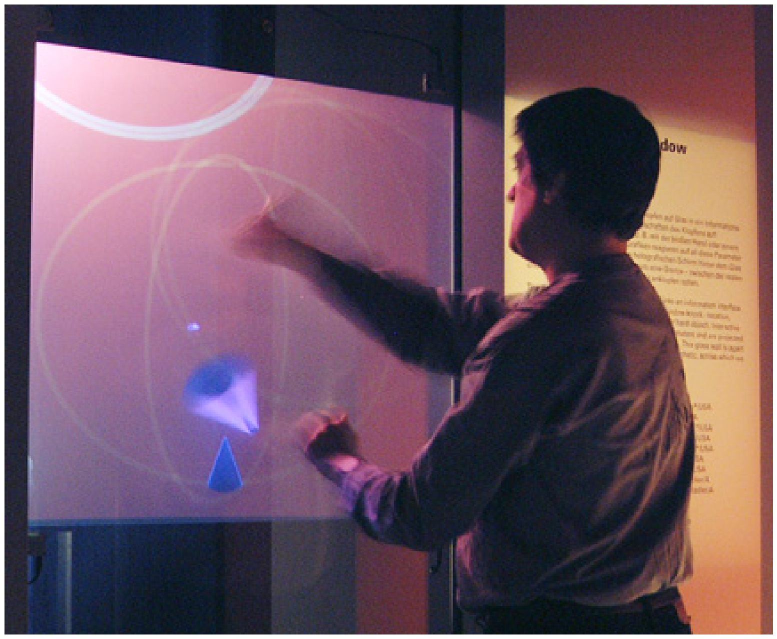
Figure 7-2: Close-up of Interactive Drawing Exhibit

A fourth knock sent the object spiraling into the screen (with rate depending on knock intensity), vanishing at the impact location. The color of the object was a function of the spectral content of the sensor waveforms, with knuckle knocks producing bluish objects and metal taps producing reddish objects. Fist bangs flash the entire screen with a bright red transient, with large rotating rings centered at the estimated impact location. Finally, hits above and below the sensor perimeters launched circular waves into the screen, centered on the assumed horizontal location of the hit.

One of the major drawbacks of this setup was that the projector had very poor illumination of the exhibit, because of surrounding ambient light coming from the windows behind the display during the daytime.

7.2 Knock-Driven Display

A knock-driven display was created on a 160 cm $\times$ 130 cm sheet of $\frac{1}{4}$ inch glass. The content augmented on the Interactive Drawing Exhibit in one particular manner: it not only projected visual images, but also integrated a Doppler radar to allow the user to somewhat control interactive sound as they moved about in front of the glass. This exhibit debuted at MIT for ID/Entity [20], and shown from December 2001 to January 2002 at "The Kitchen" in New York City. A tap at a specific location of the cover image (the artist's desktop - see Figure 7-3) allowed a story to unfold, narrated by sections of answering machine messages and accompanying video. Eight messages were mapped to different parts of the screen, allowing the particular message to be selected by knocking on a relevant object. During a video, the user could bring up images of the speaker or relevant topics by knocking on the glass. The image would appear at the knock location, and the rotational rate of the images was proportional to the user's impact strength. Figure 7-4 shows a user bringing up background images during playback.

The Doppler radar system was placed behind the top panel of the exhibit. One drawback in constructing this display window was that most of the exhibit was constructed from scratch, and the projector required a dark room with about ten feet of throw behind the screen. Because of its rear-screen nature, the exhibit had to be shown in low light conditions. The audio portions were well heard, and did not cause the system to falsely trigger. Because of the museum-based nature of the exhibit, ambient noise was relatively low.

7.3 Store Front Window

The Responsive Window System is simple, cheap, and easy to install on large windows. As nothing is needed outside the glass, it is well-matched to interactive retail on shop windows. Hence, we tested this concept by retrofitting the main window

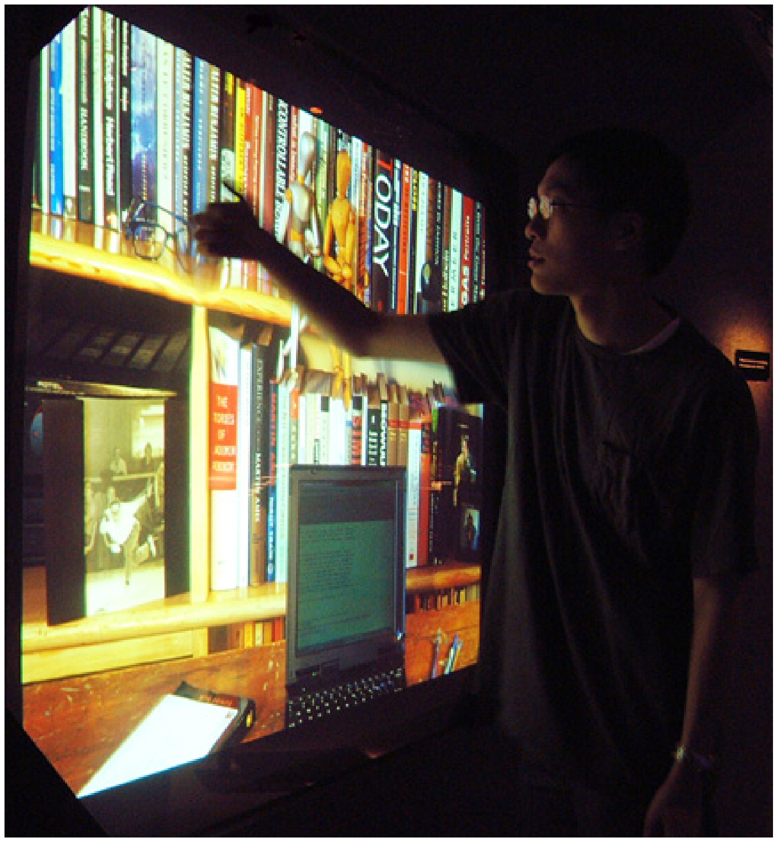

Figure 7-3: “A Telephone Story” Exhibit opening screen

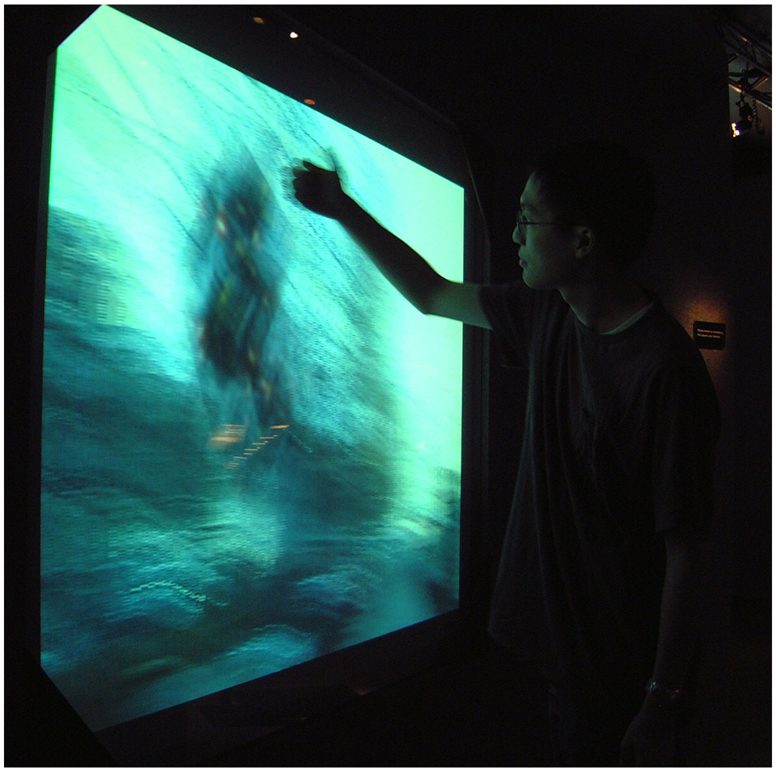

Figure 7-4: “A Telephone Story” Exhibit in action

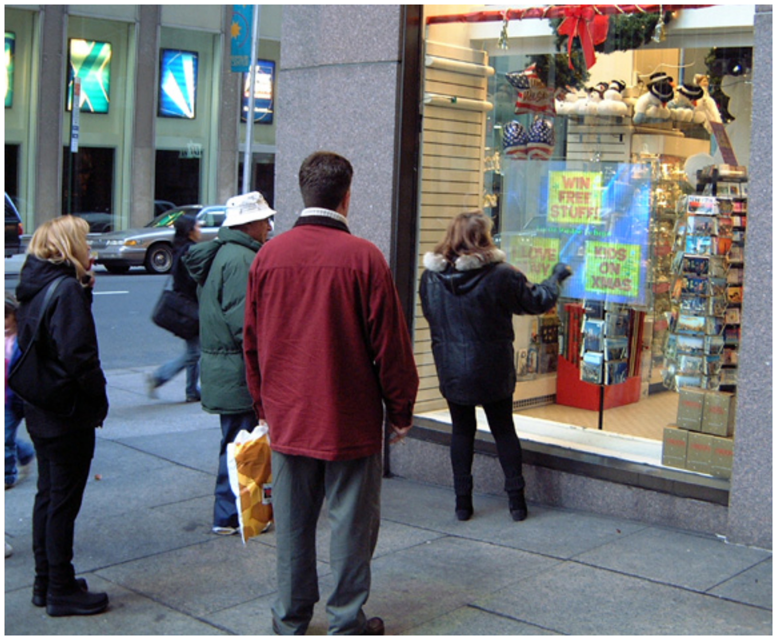

Figure 7-5: Storefront Display at American Greetings

of an American Greetings card store at Rockefeller Center in New York City for their major seasons (Christmas and Valentine's Day), running from December 2001 through February 2002. A simple interface allowed passersby to knock on the window and watch movie clips or play a card-swap game in Monte Carlo style to win a free greeting card from the store. Features from the previous two systems were used - an improved holoscreen inside the window and sound outside. Figure 7-5 shows the display window in action and Figure 7-6 shows a close-up view of the system.

One thing lacking in the system is the capability of the system to have two clap veto microphones. While a microphone inside could veto internal store noise, such as people clattering nearby merchandise, it could not veto external traffic noise and other outdoor phenomenon. Fortunately, even in downtown Manhattan, this presented only an occasional problem, and the system seldom false-triggered; this is most likely because the outside acoustic noise did not impedance-match well to the window, and thus did not frequently falsely trigger the system. The window display looked a little

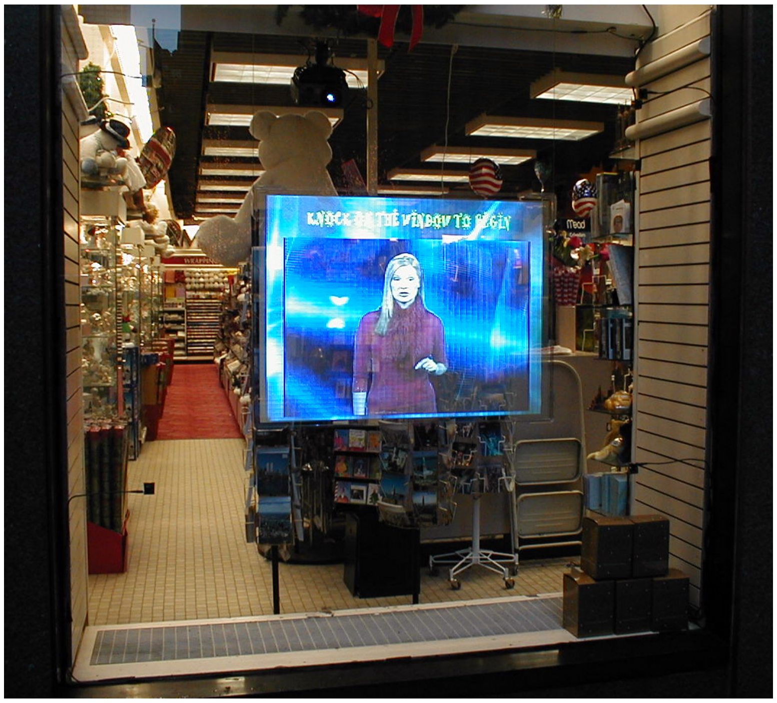

Figure 7-6: Close-up of storefront display at American Greetings

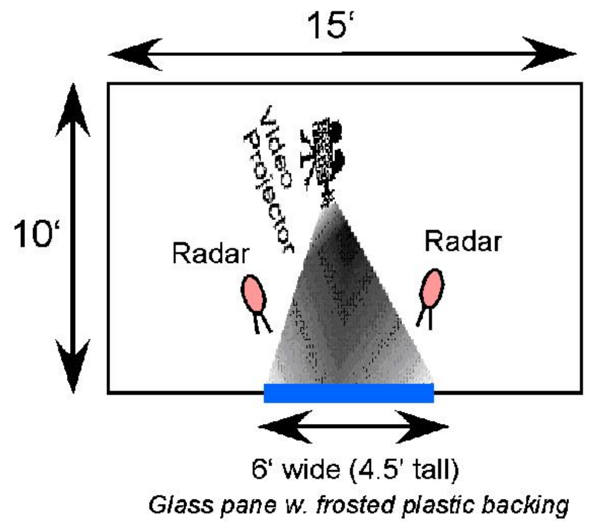

Figure 7-7: Setup for the Interactive Window at SIGGRAPH, 2002

washed out during the daytime when one was standing close, but looked brilliant at dusk. Though this store would close at night, we look forward to future trials with stores that will keep the display running throughout the evening to allow curious users to explore the content, and define a new meaning to window browsing.

7.4 SIGGRAPH: Interactive Window

A fourth application using the entire system can utilize the three dimensional sensing capabilities of the system as input to an artificial lifeform. This could be anything from a physical robot sitting behind a window, to a sentient abstract being which is projected from a computer. At SIGGRAPH 2002, this system will be unveiled [21]. Incorporating the radar and responsive window tracker, users will be able to fully explore media that both responds to contact and non-contact movements. The setup is shown in Figure 7-7.

This system will use the window parameters and radar inputs to drive a graphics system based on a layered behavioral network [22]; sample graphics are illustrated in

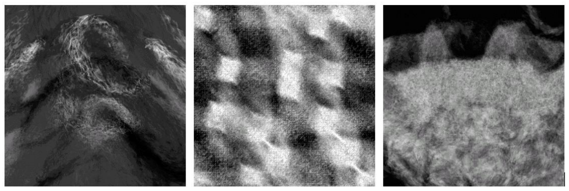

Figure 7-8: Sample graphics for Interactive Window

Figure 7-8. The exhibit intends to characterize noncontact gesture with non-localized phenomena, while contact gesture will “crystallize” discrete events at particular locations. Feedback between the user, video, and sound, will allow the Interactive Window to come alive.

Chapter 8

Conclusion and Future Work

A quick overview of the contents provided in this thesis will reveal that a stable, time-domain algorithm was developed to passively track single source, impulsive impacts. The addition of a radar system was used to provide a three dimensional interaction for the user. Several areas of research can be used to improve system robustness and portability. These include miniaturizing the components, scaling to larger windows and other media, as well as further investigation of the methods discussed in Chapter 5, such as dispersion compensation. Other types of improvements are discussed below.

8.1 Miniaturizing Components

Because this is a prototype, larger components and a DSP evaluation kit were used. As seen in Figures 3-5 and 3-6, the prototype hardware, although functional, is bulky, fragile, and complex. A next step would be to redesign the system to fit on a single smaller board or PC card. Miniaturizing the assembly of the Responsive Window system would be a quick way to increase portability and robustness of the system. Placing all the parts in a smaller package, as well as moving the serial port to a USB connection would allow the system to be more easily deployed, as well as able to interface with a wider range of computers.

It would also be possible to create a simpler device that only samples the waveforms and transmits the data over a USB port to the computer. However, in order to maintain the same response time as the DSP, we would need at least a 2.4 Mbit/s transfer rate, similar to a video system. (a rate of 2.4 Mbit/s is needed to sustain 50 kHz 12-bit sampling for four channels.)

8.2 Scaling to Larger Windows

Due to the absorption loss in glass, the Responsive Window is expected to have poor sensitivity for extremely large windows, requiring very hard knocks. A natural solution would be to add more sensors to sample more often to increase sensitivity. The algorithms described in this thesis are easily scaled to provide tracking with more sensors. However, as mentioned in Chapter 5, one must be careful about the placement of the sensors, as this can influence the sampled waveforms' characteristics.

Another method for scaling to a larger window is to use the sensor placement (2), such as discussed in Chapter 5. In order to make the algorithm work, however, the sensors will need to be placed far enough from the window edges so that reflections do not corrupt the signal. The hyperbola intersection routine can easily be modified by rotating the coordinates to maintain the conditions set to solve for the solution.

8.3 Scaling to Other Media

While these techniques have been refined for use with glass, the concepts used here could be applied to track single ping source impacts on other media. Other types of surfaces, such as plexiglass and metal sheets could be explored. However, it is possible that the receivers may have to be discarded in favor of different sensors that have better frequency responses for the given media.

8.4 Future Applications

A system that follows both contact gesture (as physical impacts) and non-contact gestures of a user has many applications. Besides being able to provide information to a user via dynamic projected video, it can also be used as a simple means of controlling devices (e.g., a vending machine or robots performing hazardous tasks) that require being behind a barrier, such as glass.

As the intrinsic resolution is so coarse, this system is mainly appropriate for large displays; standard touchscreen technologies [23] are better suited to smaller CRT-sized systems, such as in video kiosks. As a very inexpensive retrofit to large display surfaces, however, this system holds the promise of ubiquitously ushering communal and informative interaction into public venues, potentially revolutionizing our concept of window browsing.

Appendix A

Abbreviations and Symbols

Symbol	Definition
$\nabla^2 = \frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2} + \frac{\delta^2}{\delta z^2}$	Laplacian Operator
5NN	Five Nearest Neighbors
A/D	Analog to Digital
ADMC401	Analog Devices Motor Control Kit
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FMCW	Frequency Modulated Continuous Wave
IC	Integrated Circuit
MIR	Micro-Impulse Radar
PC	Personal Computer
PVDF	Polyvinylidene Flouride
SNR	Signal to Noise Ratio
USB	Universal Serial Bus

Appendix B

Hardware Schematics

This section contains schematics and layout for the Analog Components of the Responsive Window System. These include the Pre-Amplifier Board, the Main Analog Signal Conditioning Board, and the Auxiliary Power Breakout Board.

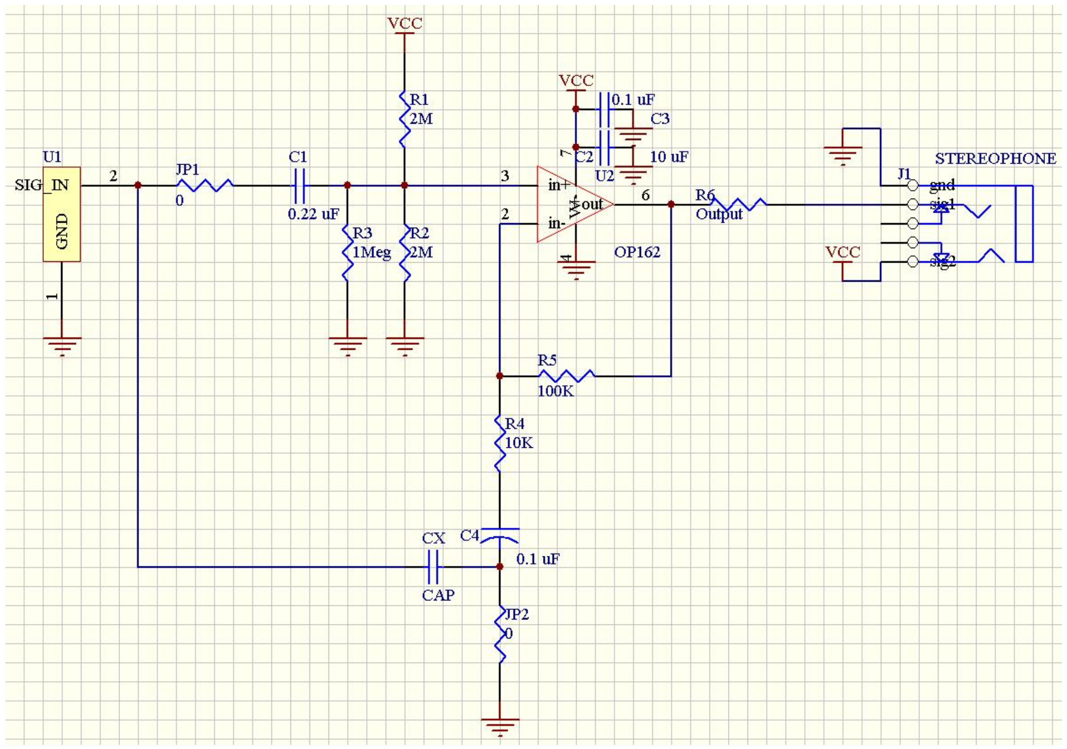

Figure B-1: Schematic for pre-amplifier board

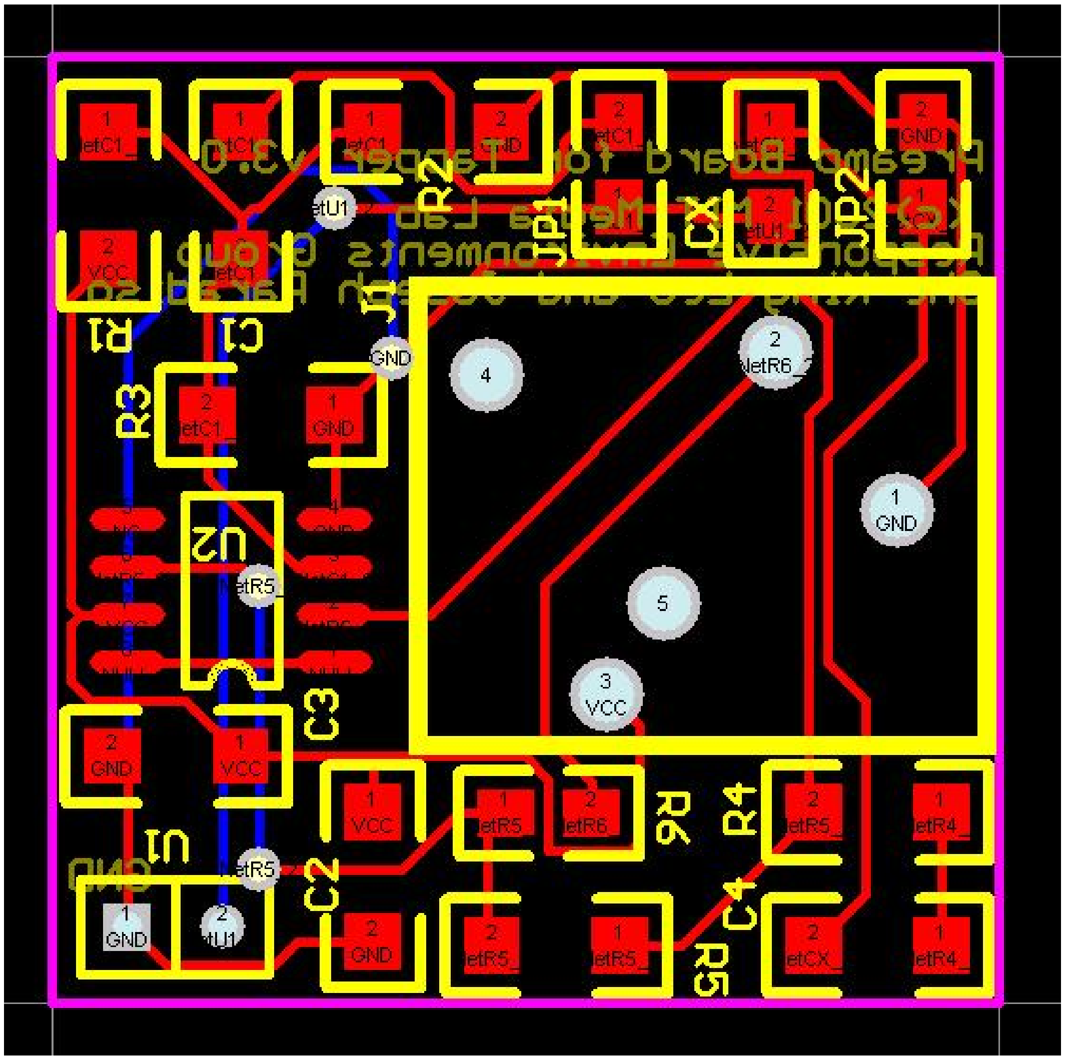

Figure B-2: Layout for pre-amplifier board

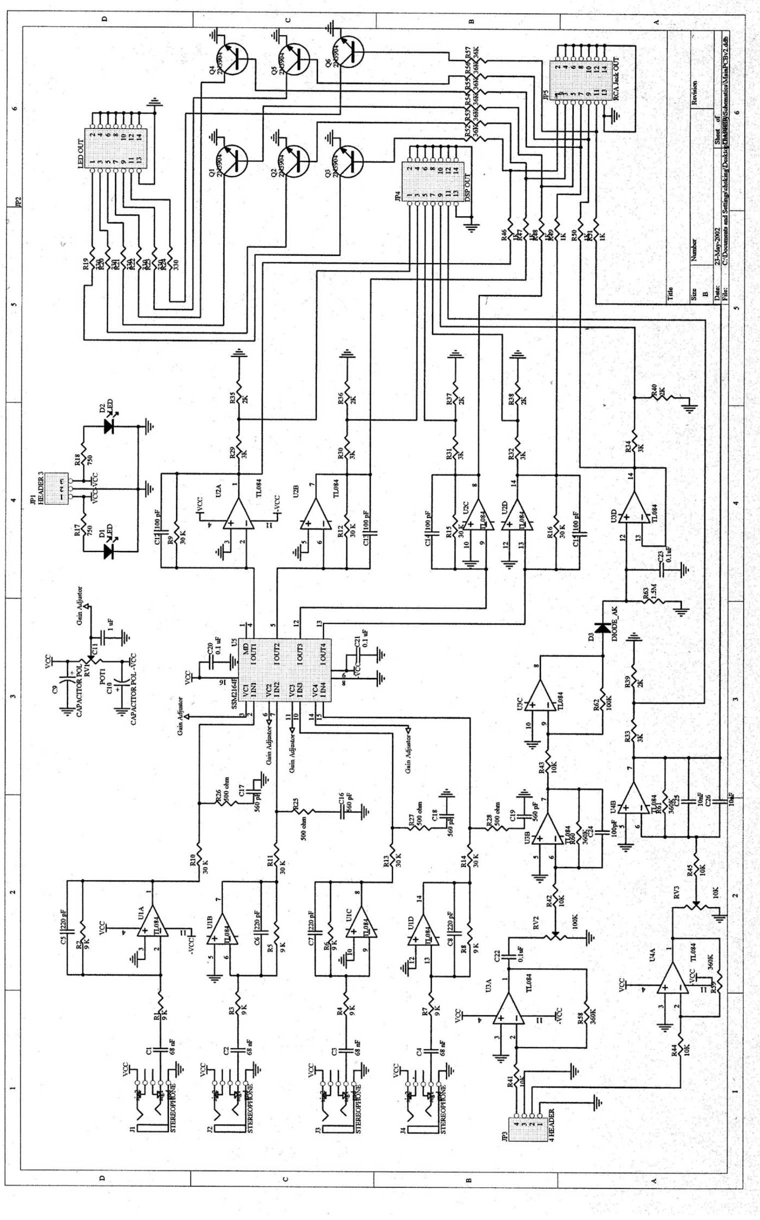

Figure B-3: Schematic for analog signal conditioning board

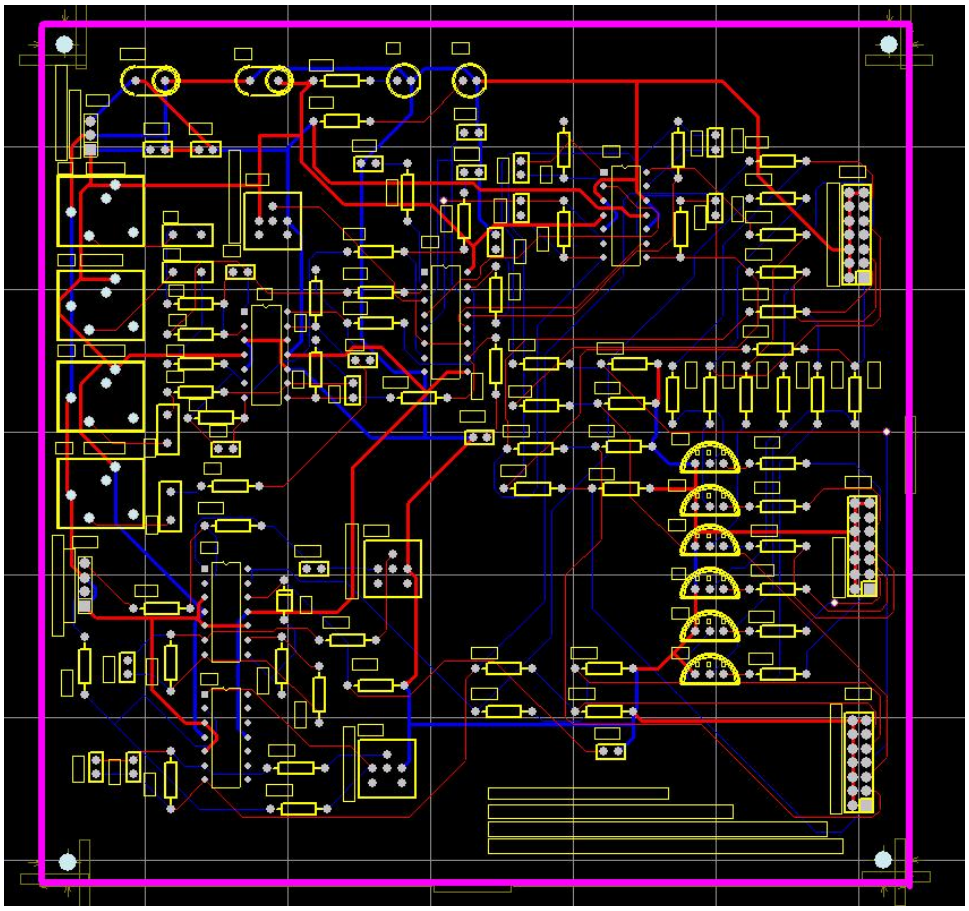

Figure B-4: Layout for analog signal conditioning board

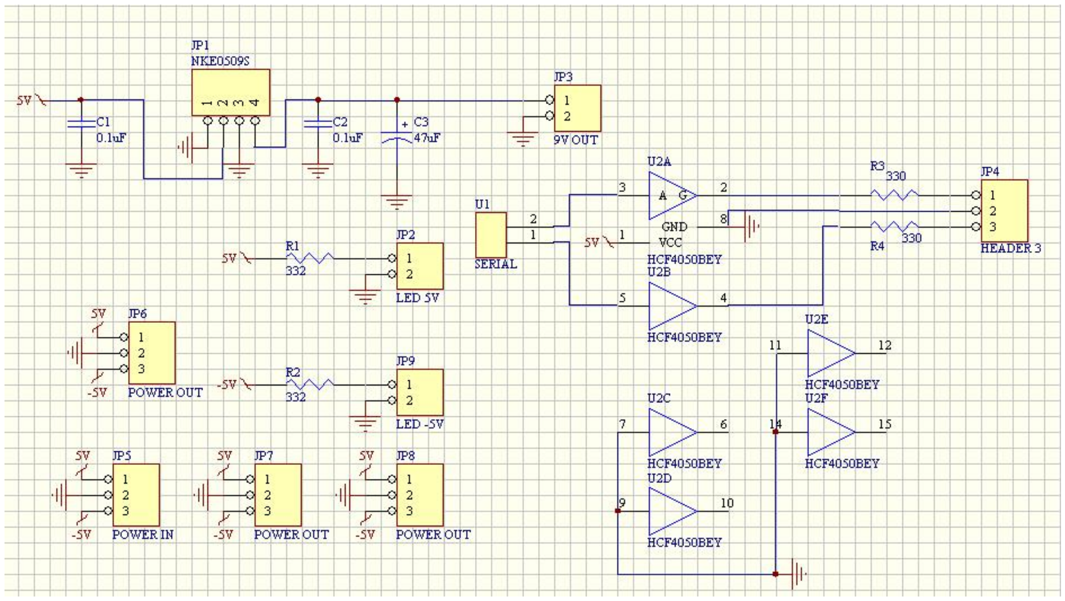

Figure B-5: Schematic for power supply auxiliary board

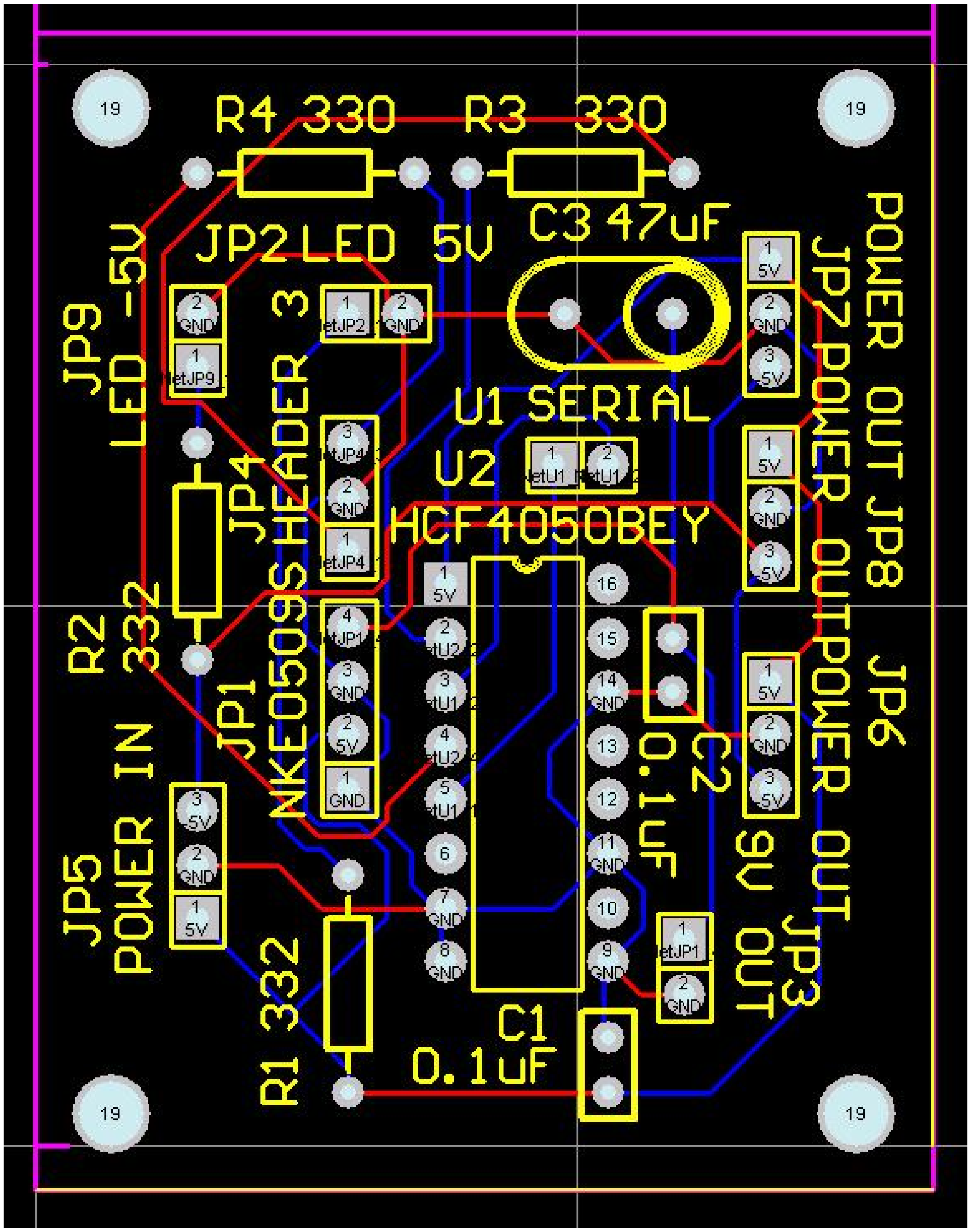

Figure B-6: Layout for power supply auxiliary board

Appendix C

Software Code

This section contains code used for the Responsive Window. It is broken up into two parts: DSP Code, and C^{++} Code.

C.1 DSP Code

main.dsp

```
{****** main.dsp *************************}
{****** Main program for the *************}
{****** Responsive Window ****************}



.MODULE/RAM/SEG=USER_PM1/ABS=0x60 Main_Test;

#include <main.h>;
#include <pwm401.h>;
#include <adc401.h>;
#include <uart0.h>;
#include <chisq.h>;

.VAR/DM/CIRC  input1[400];
.GLOBAL input1;
.VAR/DM/CIRC  input2[400];
.GLOBAL input2;
.VAR/DM/CIRC  input3[400];
.VAR/DM/CIRC  input4[400];

.VAR/DM input1save[400];
.VAR/DM input2save[400];
.VAR/DM input3save[400];
.VAR/DM input4save[400];
.VAR/DM inputsaveptr;
.VAR/DM micsave[1000];
.VAR/DM mic2save[400];

.VAR/DM inp1start13;
.VAR/DM inp2start13;
.VAR/DM inp1end13;
.VAR/DM inp2end13;
.VAR/DM maxcorrdiff13;
.VAR/DM secondcorrdiff13;

.VAR/DM/CIRC  mic[1000];
.VAR/DM/CIRC  mic2[400];
.VAR/DM tempinput[400];
.GLOBAL tempinput;
.VAR/DM length;

.VAR/DM xcorrhi[400];
.GLOBAL xcorrhi;
.VAR/DM xcorrlo[400];
.GLOBAL xcorrlo;
.VAR/DM locptr;
.GLOBAL locptr;
.VAR/DM peakcount;

.VAR/DM initialbias;
.VAR/DM initialsumhi;
.VAR/DM initialsumlo;
.VAR/DM tapthres;
.VAR/DM bangthres;
.VAR/DM delaybeforenext;
.VAR/DM delaycntr;
.VAR/DM wasbadtap;
.VAR/DM metaltapthres;
.VAR/DM toolonginmain;
.VAR/DM abovenoisethres;
.VAR/DM noisecount;
.VAR/DM maxsum;
.VAR/DM littledelaycntr;
.VAR/DM littledelaymult;
.VAR/DM pwmtripcount;

.VAR/DM inp1value;
.VAR/DM inp2value;
.VAR/DM inp3value;
.VAR/DM inp4value;
.VAR/DM inp5value;
.VAR/DM inp6value;

.VAR/DM decaythres;
.VAR/DM decaymult;
.VAR/DM largestmax;
.VAR/DM startsamplecount;
.VAR/DM bangcount;

.VAR/DM filtersumlo;
.VAR/DM filtersumhi;

.VAR/DM maxcorrpeakhi;
.VAR/DM maxcorrpeaklo;
.VAR/DM maxcorrpeakloc;
.VAR/DM secondcorrpeakhi;
.VAR/DM secondcorrpeaklo;
.VAR/DM secondcorrpeakloc;
.VAR/DM maxcorrdiff;
.VAR/DM secondcorrdiff;

.VAR/DM corrlength;
.GLOBAL corrlength;
.VAR/DM peakamphi[15];
.VAR/DM peakamplo[15];
.VAR/DM peakloc[15];

.VAR/DM zerocrossloc[15];
.VAR/DM zerocrosscount;
.VAR/DM purezerocrosscount;
.VAR/DM max;
.GLOBAL max;
.VAR/DM threshhold;
.GLOBAL threshhold;
.VAR/DM min;
.GLOBAL min;
.VAR/DM sum;
.VAR/DM sumlength;
.VAR/DM sumcounter;
.VAR/DM trimstart;
.GLOBAL trimstart;
.VAR/DM zerocross;
.VAR/DM maxpeak;
.VAR/DM inputlabel;
.VAR/DM aftertriggerlength;
.VAR/DM saveinput;
.VAR/DM saveinput1;
.VAR/DM saveinput2;
.VAR/DM saveinput3;
.VAR/DM hpsum;
.VAR/DM slopepos1;
.VAR/DM slopepos2;
.vAR/DM slopeneg1;
.VAR/DM slopeneg2;
.VAR/DM peakorzero;
.VAR/DM lastpeakfound;
.VAR/DM notmaxpeak;
.VAR/DM searchdownslope;
.VAR/DM prevslope;
.VAR/DM ninetenthsmax;
.VAR/DM lastpeakamp;
.VAR/DM currpeakamp;

.VAR/DM threepeaks[3];
.VAR/DM surrzeros[4];
.VAR/DM threepeaks1[3];
.VAR/DM threepeaks2[3];
.VAR/DM threepeaks3[3];
.VAR/DM threepeaks4[3];
.VAR/DM surrzeros1[4];
.VAR/DM surrzeros2[4];
.VAR/DM surrzeros3[4];
.VAR/DM surrzeros4[4];

.VAR/DM inp1peaksptr;
.GLOBAL inp1peaksptr;
.VAR/DM inp2peaksptr;
.GLOBAL inp2peaksptr;
.VAR/DM inp1surrzerosptr;
.GLOBAL inp1surrzerosptr;
.VAR/DM inp2surrzerosptr;
.GLOBAL inp2surrzerosptr;
.VAR/DM inp1start;
.GLOBAL inp1start;
```

```
.VAR/DM inp1end;
.GLOBAL inp1end;
.VAR/DM inp2start;
.GLOBAL inp2start;
.vAR/DM inp2end;
.GLOBAL inp2end;
.VAR/DM inp1start2;
.GLOBAL inp1start2;
.VAR/DM inp1end2;
.GLOBAL inp1end2;
.VAR/DM inp2start2;
.GLOBAL inp2start2;
.VAR/DM inp2end2;
.GLOBAL inp2end2;
.VAR/DM inp1num;
.GLOBAL inp1num;
.VAR/DM inp2num;
.GLOBAL inp2num;
.VAR/DM inp1startfinal;
.GLOBAL inp1startfinal;
.VAR/DM inp1endfinal;
.GLOBAL inp1endfinal;
.VAR/DM inp2startfinal;
.GLOBAL inp2startfinal;
.VAR/DM inp2endfinal;
.GLOBAL inp2endfinal;
.VAR/DM finalchisqminlo;
.GLOBAL finalchisqminlo;
.VAR/DM finalchisqminhi;
.GLOBAL finalchisqminhi;
.VAR/DM inp1ptr;
.GLOBAL inp1ptr;
.VAR/DM inp2ptr;
.GLOBAL inp2ptr;

.VAR/DM chisqinput1[100];
.GLOBAL chisqinput1;
.VAR/DM chisqinput2[100];
.GLOBAL chisqinput2;
.VAR/DM chisqlen1;
.GLOBAL chisqlen1;
.VAR/DM chisqlen2;
.GLOBAL chisqlen2;
.VAR/DM chisqmaxlen;
.GLOBAL chisqmaxlen;
.VAR/DM chisqminlo;
.GLOBAL chisqminlo;
.vAR/DM chisqminhi;
.GLOBAL chisqminhi;
.VAR/DM checkdist;
.GLOBAL checkdist;
.VAR/DM newmin;
.GLOBAL newmin;
.VAR/DM shiftinput;
.GLOBAL shiftinput;
.VAR/DM mulfacthi;
.GLOBAL mulfacthi;
.VAR/DM mulfactlo;
.GLOBAL mulfactlo;

.VAR/DM maxminlength;
.GLOBAL maxminlength;

.VAR/DM metaltapstart;
.VAR/DM zerocrosssum;

.VAR/DM hextemp;

.VAR/DM startdata;
.vAR/DM trigger;

.ENTRY findmaxmin;
.ENTRY badtap;

{************************}
{ Start of program code }
{************************}

startup:
    IMASK=0x0006;
    AR=0;
    dm(trigger)=AR;
    dm(startdata)=AR;

    PWM_Init(PWMSYNC_ISR, PWMTRIP_ISR);

    UART0_Init;
```

```
    IFC = 0x80;              { Clear any pending IRQ2 inter.}
    IMASK=0x0206;

    ADC_Init;
            { Calibrates the ADC block. This calibration requires }
            { values from the ADC and so the PWMSYNC must be  }
            { running when it is called. Here all the offset are }
            { stored.                          }
            { Thus, ADC_init is placed after IRQ2 is enabled  }
    IMASK = 0x0006;

    call initvalues;

    AR=0;
    dm(pwmtripcount)=AR;

    IMASK = 0x0206;

    jump main;

initvalues:
    AR=%input1;
    dm(length) = AR;

    AR=251;
    dm(aftertriggerlength)=AR;

    AR=120;
    dm(sumlength)=AR;

    AR=10;
    dm(checkdist)=AR;

    AR=0x2000; {was 0x1600}
    dm(tapthres)=AR;

    AR=0x6000;
    dm(bangthres)=AR;

    AR=5000; {was 600}
    dm(delaybeforenext)=AR;

    AR=1;
    dm(delaycntr)=AR;

    AR=0;
    dm(wasbadtap)=AR;

    AR=42;
    dm(metaltapthres)=AR;

    AR=0x0050;
    dm(abovenoisethres)=AR;

    AR=40;
    dm(noisecount)=AR;

    AR=0;
    dm(littledelaycntr)=AR;

    AR=150;
    dm(littledelaymult)=AR;

    AR=0;
    dm(decaythres)=AR;

    AR=0x0001;
    dm(decaymult)=AR;

    AR=-1;
    dm(bangcount)=AR;

    AR=0;
    dm(trigger)=AR;

    AR=150;
    dm(startsamplecount)=AR;

    write_dm(piopwm, 0);

    I1=^input1;
    L1=%input1;
    M1=1;
    I2=^input2;
    L2=L1;
    M2=1;
    I3=^input3;
    L3=L1;
    M3=1;
    I4=^input4;
    L4=L1;
    M4=1;
    I0=^mic;
    L0=%mic;
    M0=1;
    I5=^mic2;
    L5=%mic2;
    M5=1;

    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine1;
    AR=1;
cntrfine1: CNTR=AR;
    AR=0;

    DO init1 UNTIL CE;
        DM(I1, M1)=AR;
        DM(I2, M2)=AR;
        DM(I3, M3)=AR;
        DM(I4, M4)=AR;
        DM(I0, M0)=AR;
init1:     DM(I5, M5)=AR;

    AR=%mic;
    CNTR=AR;
    AR=0;
    DO initmic UNTIL CE;
initmic:   DM(I0, M0)=AR;

    AR=%peakamphi;
    CNTR=AR;
    AR=0;

    I5=^peakamphi;
    I6=^peakamplo;
    I7=^peakloc;
    L5=0;
    L6=0;
    L7=0;
    M5=1;
    M6=1;
    M7=1;

    DO init2 UNTIL CE;
        DM(I5, M5)=AR;
        DM(I6, M6)=AR;
init2:     DM(I7, M7)=AR;

    I5=^threepeaks;
    AR=3;
    CNTR=AR;
    AR=0;
    DO init3 UNTIL CE;
init3: DM(I5, M5)=AR;

    AR=1;
    dm(startdata)=AR;

    call resetinputs;

RTS;


MAIN:
    nop;
    nop;
    nop;
    nop;
    nop;
    nop;
    nop;
    jump MAIN;
rts;

testthreepeaks:
    IMASK=0x0006;
    I6=AR;
    M6=1;
    L6=0;
    AR=dm(I6, M6);
    AR=AR+0;
    IF GT jump okay;
    AR=0x1234;
okay:  nop;
RTS;

save13info:
    IMASK=0x0006;
    AR=dm(inp1startfinal);
    dm(inp1start13)=AR;
    AR=dm(inp2startfinal);
    dm(inp2start13)=AR;
    AR=dm(inp1endfinal);
    dm(inp1end13)=AR;
    AR=dm(inp2endfinal);
    dm(inp2end13)=AR;
    AR=dm(maxcorrdiff);
    dm(maxcorrdiff13)=AR;
    AR=dm(secondcorrdiff);
    dm(secondcorrdiff13)=AR;
RTS;

printdigit: {put digit to be printed in AR}
    IMASK=0x0066;

    ax0=AR;
    ay0=0xA;
    AR=AR-ay0;
    if GE jump letter;
        ay1=0x30;
        AR=ax0;
        AR=AR+ay1;
        UART0_Write(AR);
        jump endprint;
letter:
    AR=ax0;
    AR=AR-ay0;
    ay1=0x41;
    AR=AR+ay1;
    UART0_WRite(AR);
endprint:  nop;

RTS;

printhex: {put number to be printed in AR}
    IMASK=0x0066;

    dm(hextemp)=AR;

    ay0=0xF000;
    AR=AR AND ay0;
    SR=LSHIFT AR BY -12 (hi);
    AR=SR1;
    call printdigit;

    AR=dm(hextemp);
    ay0=0x0F00;
    AR=AR AND ay0;
    SR=LSHIFT AR BY -8 (hi);
    AR=SR1;
    call printdigit;

    AR=dm(hextemp);
    ay0=0x00F0;
    AR=AR AND ay0;
    SR=LSHIFT AR BY -4 (hi);
    AR=SR1;
    call printdigit;

    AR=dm(hextemp);
    ay0=0x000F;
    AR=AR AND ay0;
    call printdigit;

    Uart0_Write(255);
RTS;

resetinputs:
    IMASK=0x0006;
    AR=^input1;
    I1=AR;
    AR=^input2;
    I2=AR;
    AR=^input3;
    I3=AR;
    AR=^input4;
    I4=AR;
    AR=dm(length);
    L1=AR;
    L2=L1;
    L3=L1;
    L4=L1;
    M1=1;
    M2=1;
    M3=1;
    M4=1;
    I0=^mic;
    L0=%mic;
    M0=1;
    I5=^mic2;
    L5=%mic2;
    M5=1;
RTS;


copyinputs: {recopy 4 inputs as linear arrays}
    IMASK=0x0006;

    I6=^tempinput;
    L6=0;
    M6=1;

    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine12;
    AR=1;
cntrfine12: CNTR=AR;
    DO copyloop1 UNTIL CE;
        AR=dm(I1, M1);
copyloop1: dm(I6, M6)=AR;
    I1=^input1;
    L1=0;
    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine2;
    AR=1;
cntrfine2: CNTR=AR;
    DO copyloop2 UNTIL CE;
        AR=dm(I6, M6);
copyloop2: dm(I1, M1)=AR;

    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine3;
    AR=1;
cntrfine3: CNTR=AR;
    DO copyloop3 UNTIL CE;
        AR=dm(I2, M2);
copyloop3: dm(I6, M6)=AR;
    I2=^input2;
    L2=0;
    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine4;
    AR=1;
cntrfine4: CNTR=AR;
    DO copyloop4 UNTIL CE;
        AR=dm(I6, M6);
copyloop4: dm(I2, M2)=AR;

    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine5;
    AR=1;
cntrfine5: CNTR=AR;
    DO copyloop5 UNTIL CE;
        AR=dm(I3, M3);
copyloop5: dm(I6, M6)=AR;
    I3=^input3;
    L3=0;
    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine6;
    AR=1;
cntrfine6: CNTR=AR;
    DO copyloop6 UNTIL CE;
        AR=dm(I6, M6);
copyloop6: dm(I3, M3)=AR;

    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine7;
    AR=1;
cntrfine7: CNTR=AR;
    DO copyloop7 UNTIL CE;
        AR=dm(I4, M4);
copyloop7: dm(I6, M6)=AR;
    I4=^input4;
    L4=0;
    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine8;
    AR=1;
cntrfine8: CNTR=AR;
    DO copyloop8 UNTIL CE;
        AR=dm(I6, M6);
copyloop8: dm(I4, M4)=AR;

    I6=^tempinput;
    AR=%mic;
    AR=AR+0;
    IF GT jump cntrfine9;
    AR=1;
cntrfine9: CNTR=AR;
    DO copyloop9 UNTIL CE;
        AR=dm(I0, M0);
copyloop9: dm(I6, M6)=AR;
    I0=^mic;
    L0=0;
    I6=^tempinput;
    AR=%mic;
    AR=AR+0;
    IF GT jump cntrfine10;
    AR=1;
cntrfine10: CNTR=AR;
    DO copyloop10 UNTIL CE;
        AR=dm(I6, M6);
copyloop10: dm(I0, M0)=AR;

    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine11;
    AR=1;
cntrfine11: CNTR=AR;
    DO copyloop11 UNTIL CE;
        AR=dm(I5, M5);
copyloop11: dm(I6, M6)=AR;
    I5=^mic2;
    L5=0;
    I6=^tempinput;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine13;
    AR=1;
cntrfine13: CNTR=AR;
    DO copyloop12 UNTIL CE;
        AR=dm(I6, M6);
copyloop12: dm(I5, M5)=AR;

RTS;

findinitialbias:
    IMASK=0x0006;

    L5=0;
    M5=1;
    AR=0;
    dm(initialsumlo)=AR;
    dm(initialsumhi)=AR;
    AR=8;
    CNTR=AR;
    DO suminitial UNTIL CE;
        ay0=dm(I5, M5);
        ax0=10000;
        AR=ax0+ay0;
        ay0=dm(initialsumlo);
        AR=AR+ay0;
        dm(initialsumlo)=AR;
        AR=dm(initialsumhi);
        AR=AR+C;
suminitial: dm(initialsumhi)=AR;
    AR=dm(initialsumlo);
    SR=LSHIFT AR BY -3 (lo);
    AR=dm(initialsumhi);
    SR=SR OR LSHIFT AR BY -3 (hi);
    AR=SR0;
    ay0=10000;
    AR=AR-ay0;
    dm(initialbias)=AR;
RTS;


findmaxmin:
    {put sample in I5, length in maxminlength}
    IMASK=0x0006;
    L5=0;
    M5=1;
    AR=0x7FFF;
    ay0=AR;
    AR=0x0000;
    ay1=AR;
    AR=dm(maxminlength);
    AR=AR+0;
    IF LE RTS;
    CNTR=AR;

    DO maxloop UNTIL CE;
        ax1=dm(I5, M5);
        AR=ax1+0;
        IF GT jump posalready;
        AR=-ax1;
        ax1=AR;
posalready: AR=ax1-ay0;
        IF GT jump checkmax;
        ay0=ax1;
checkmax:  AR=ax1;
        AR=AR-ay1;
        IF LE jump maxloop;
        ay1=ax1;
maxloop:   nop;

    AR=ay0;
    dm(min)=AR;
    AR=ay1;
    dm(max)=AR;
RTS;


normalize:
    {put sample in I5, put savesample in I7}
    IMASK=0x0006;

    M5=1;
    L5=0;
    I6=I5;
    M6=1;
    L6=0;

    M7=1;
    L7=0;

    AR=80;
    dm(maxminlength)=AR;
    call findmaxmin;

    I5=I6;
    M5=1;
    L5=0;

    AR=1;
    dm(purezerocrosscount)=AR;

    AR=dm(max);
    ay0=1600;
    AR=AR+ay0;
    dm(max)=AR;
    AR=dm(min);
    AR=AR-ay0;
    dm(min)=AR;

    AR=0;
    dm(abovethreshhold)=AR;

    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrisfine;
        AR=1;
cntrisfine: CNTR=AR;
    DO updatezerocount UNTIL CE;
        AR=dm(abovethreshhold);
        AR=AR+0;
        IF NE jump lookforzeroxing;
            AR=dm(I5, M5);
            ay0=AR;
            AR=dm(max);
            AR=AR-ay0;
            IF LE jump abovethresh;
                AR=dm(min);
                AR=AR-ay0;
                IF LT jump updatezerocount;
                    AR=-1;
                    dm(abovethreshhold)=AR;
                    jump updatezerocount;
abovethresh:        AR=1;
            dm(abovethreshhold)=AR;
            jump updatezerocount;
lookforzeroxing:AR=AR+0;
        IF GT jump lookdownwards;
            AR=dm(I5, M5);
            ay0=AR;
            AR=dm(max);
            AR=AR-ay0;
            IF GT jump updatezerocount;
                AR=dm(purezerocrosscount);
                AR=AR+1;
                dm(purezerocrosscount)=AR;
                AR=1;
                dm(abovethreshhold)=AR;
                jump updatezerocount;
lookdownwards: AR=dm(I5, M5);
        ay0=AR;
        AR=dm(min);
        AR=AR-ay0;
        IF LT jump updatezerocount;
            AR=dm(purezerocrosscount);
            AR=AR+1;
            dm(purezerocrosscount)=AR;
            AR=-1;
            dm(abovethreshhold)=AR;
updatezerocount:nop;

    I5=I6;
    AR=1600;
    ay0=AR;
    AR=dm(max);
    AR=AR-ay0;
    dm(max)=AR;
    AR=dm(min);
    AR=AR+ay0;
    dm(min)=AR;


    ay0=dm(initialbias);
    AR=dm(max);
    AR=AR-ay0;

    dm(max)=AR;

    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine14;
    AR=1;
cntrfine14: CNTR=AR;

    AR=dm(inputlabel);
    ay1=5;
    AR=AR-ay1;
    IF NE jump notfive2;

    AR=%mic;
    AR=AR+0;
    IF GT jump cntrfine30;
    AR=1;
cntrfine30: CNTR=AR;

notfive2:  AR=dm(I5, M5);
    DO submin UNTIL CE;
        AR=AR-ay0;
        IF GT jump waspositive;
        AR=-AR;
waspositive:  SR=LSHIFT AR BY -4 (LO);
        AR=SR0;
        dm(I6, M6)=AR;
        dm(I7, M7)=AR;
submin:    AR=dm(I5, M5);

    AR=dm(max);
    SR=LSHIFT AR BY -4 (LO);
    dm(max)=SR0;
    SR=LSHIFT AR BY -8 (lo);
    AR=SR0;
    dm(threshhold)=AR;
RTS;

trimfront: {put sample in I5}
    IMASK=0x0006;

    M5=1;
    L5=0;
    I6=I5;
    M6=1;
    L6=0;
    AR=0;
    dm(trimstart)=AR;

    ay0=dm(threshhold);
    ax0=15;
    AR=ax0-ay0;
    IF LE jump takethres;
    ay0=15;
takethres: AR=1;
    dm(locptr)=AR;
    AR=dm(length);
    AR=AR+0;
    IF GT jump cntrfine15;
    AR=1;
cntrfine15: CNTR=AR;
    AR=dm(I5, M5);
    DO trimloop UNTIL CE;
        AR=AR-ay0;
        IF LT JUMP notreached;
        AR=dm(locptr);
        dm(trimstart)=AR;
        pop CNTR;
        pop PC;
        pop LOOP;
        jump trimloopdone;
notreached: AR=0;
        dm(I6, M6)=AR;
        AR=dm(locptr);
        AR=AR+1;
        dm(locptr)=AR;
        AR=dm(I5, M5);
trimloop:  nop;
trimloopdone: nop;
RTS;

findpeakzero: {put input in I5}
    IMASK=0x0006;

    AR=dm(trimstart);
    AR=AR+0;
    IF LE jump trimstartzero;
    AR=AR-1;
trimstartzero: ay0=AR;
    AR=I5;
    dm(saveinput)=AR;
    AR=AR+ay0;
    I5=AR;

    I6=^peakloc;
    I7=^zerocrossloc;
    M6=1;
    M7=0;
    L6=0;
    L7=0;
    M0=1;

    AR=0;
    dm(maxpeak)=AR;
    dm(zerocross)=AR;
    dm(sum)=AR;
    dm(slopepos1)=AR;
    dm(slopepos2)=AR;
    dm(slopeneg1)=AR;
    dm(slopeneg2)=AR;
    dm(lastpeakfound)=AR;
    dm(searchdownslope)=AR;
    dm(lastpeakamp)=AR;

    AR=1;
    dm(peakorzero)=AR;
    dm(zerocrosscount)=AR;
    dm(sumcounter)=AR;

    AR=3;
    dm(locptr)=AR;

    AR=dm(length);
    AR=AR-ay0;
    AR=AR-1;
    IF GT jump cntrokay1;
        call badtap;
        jump endloop;
cntrokay1: CNTR=AR;
    AR=dm(I5, M5);
    SR=LSHIFT AR by -4 (lo);
    AR=SR0;
    dm(sum)=AR;

    AR=dm(trimstart);
    dm(I7, M7)=AR;
    AR=dm(I5, M5);
    dm(prevslope)=AR;
    ax0=AR;

    DO pzloop UNTIL CE;

        ay0=dm(sumlength);
        AR=dm(sumcounter);
        AR=AR-ay0;
        IF LT jump sumnext;
        pop PC;
        pop LOOP;
        pop CNTR;
        jump pzloop;
sumnext:   AR=ax0;
        SR=LSHIFT AR by -4 (lo);
        ay0=dm(sum);
        AR=SR0+ay0;
        dm(sum)=AR;
        AR=dm(sumcounter);
        AR=AR+1;
        dm(sumcounter)=AR;

        ax1=dm(I5, M5);
        ay1=ax0;

        AR=dm(peakorzero);
        AR=AR+0;
        IF GT jump peaks;

        AR=dm(slopeneg2);
        AR=AR+0;
        IF LE jump notneg2;

        AR=ax1-ay1;
        IF LE jump notzero;

        AR=0;
        dm(slopeneg1)=AR;
        dm(slopeneg2)=AR;
        jump iszero;

notneg2:      AR=dm(slopeneg1);
        AR=AR+0;
        IF LE jump notneg1;
        AR=ax1-ay1;
        IF EQ jump incptr;
        IF GT jump pos1;
        AR=1;
        dm(slopeneg2)=AR;
        jump notzero;
pos1:     AR=0;
        dm(slopeneg1)=AR;
        jump notzero;

notneg1:      AR=ax1-ay1;
        IF GE jump incptr;
        AR=1;
        dm(slopeneg1)=AR;
        jump notzero;

iszero:   AR=dm(zerocrosscount);
        AR=AR+1;
        dm(zerocrosscount)=AR;

        ay0=%zerocrossloc;
        AR=dm(zerocrosscount);
        AR=AR-ay0;
        IF GE jump incptr;

        AR=1;
        dm(peakorzero)=AR;

        AR=dm(trimstart);
        AR=AR-2;
        ay0=AR;
        AR=dm(locptr);
        AR=AR+ay0;
        dm(I7, M7)=AR;
        jump incptr;

notzero:    {AR=0;
        dm(zerocross)=AR;}
        AR=dm(lastpeakfound);
        AR=AR+0;
        IF GT jump incptr;

        AR=dm(searchdownslope);
        AR=AR+0;
        IF EQ jump incptr;

        AR=ax1-ay1;
        ay0=dm(prevslope);
        AR=AR-ay0;
        AR=AR-2;
        IF LE jump incptr;
        AR=1;
        dm(lastpeakfound)=AR;

        AR=I6;
        AR=AR-2;
        I1=AR;

        AR=I7;
        AR=AR-1;
        I2=AR;

        M1=-1;
        M2=-1;
        L1=0;
        L2=0;

        I3=^threepeaks;
        I4=^surrzeros;
        M3=1;
        M4=1;

        AR=3;
        CNTR=AR;
        DO setthreepeaks UNTIL CE;
            AR=dm(I1, M1);
            dm(I3, M3)=AR;
            AR=dm(I2, M2);
setthreepeaks:    dm(I4, M4)=AR;
        AR=dm(I2, M2);
        dm(I4, M4)=AR;

        jump incptr;

peaks:      AR=dm(peakcount);
        AR=AR+0;
        IF GT jump notfirstpeak;

        AR=ax1-ay1;
        ay0=dm(prevslope);

        AR=AR-ay0;
        AR=AR-1;
        IF LE jump notfirstpeak;

        AR=dm(trimstart);
        AR=AR-2;
        ay0=AR;
        AR=dm(locptr);
        AR=AR+ay0;
        dm(I7, M7)=AR;

notfirstpeak:  AR=dm(slopepos2);
        AR=AR+0;
        IF LE jump not2;
        AR=ax1-ay1;
        IF GE jump incptr;

        AR=dm(peakcount);
        AR=AR-1;
        IF GE jump donesetptr;
        M7=1;
        AR=dm(I7, M7);

donesetptr: AR=0;
        dm(slopepos2)=AR;
        dm(slopepos1)=AR;

        AR=%peakloc;
        ay0=AR;
        AR=dm(peakcount);
        AR=AR-ay0;
        IF GT jump incptr;

        AR=dm(peakcount);
        AR=AR+1;
        dm(peakcount)=AR;

        AR=0;
        dm(peakorzero)=AR;
        dm(zerocross)=AR;

        AR=dm(trimstart);
        AR=AR-2;
        ay0=AR;
        AR=dm(locptr);
        AR=AR+ay0;
        dm(I6, M6)=AR;

        AR=dm(maxpeak);
        AR=AR-ay1;
        IF GE jump notnewmax;
        AR=ay1;
        dm(maxpeak)=AR;
        AR=0;
        dm(notmaxpeak)=AR;

        jump donesetmax;

notnewmax: AR=1;
        dm(notmaxpeak)=AR;

donesetmax: ay0=dm(peakcount);
        AR=3;
        AR=AR-ay0;
        IF LT jump morethanthree;
        AR=dm(lastpeakamp);
        AR=AR-ay1;
        IF LE jump setlastpeak;

        AR=dm(lastpeakfound);
        AR=AR+0;
        IF GT jump pzloop;

        AR=1;
        dm(lastpeakfound)=AR;

        ay0=5;
        ax0=dm(inputlabel);
        AR=ax0-ay0;
        IF GE jump pzloop;

        I1=I6;
        I2=I7;
        M1=-1;
        M2=-1;
        L1=0;
        L2=0;

        I3=^threepeaks;
        I4=^surrzeros;
        M3=1;
        M4=1;
        L3=0;
        L4=0;

        AR=dm(peakcount);
        AR=AR-1;

        IF GT jump enoughpeaks3;
            call badtap;
            jump endloop;
enoughpeaks3: AR=dm(I1, M1);
        AR=dm(I1, M1);
        AR=dm(I2, M2);
        AR=3;
        ay0=dm(peakcount);
        AR=AR-ay0;
        IF GT jump setcntrto3b;
        AR=ay0;
        AR=AR-1;
        IF GT jump cntrfine16a;
        call badtap;
        POP PC;
        POP CNTR;
        POP LOOP;
        jump endloop;
cntrfine16a:  CNTR=AR;
        jump donesettingcntr1b;
setcntrto3b:  CNTR=3;
donesettingcntr1b:
            DO setthreepeaksend1a UNTIL CE;
            AR=dm(I1, M1);
            dm(I3, M3)=AR;
            AR=dm(I2, M2);
setthreepeaksend1a: dm(I4, M4)=AR;
        AR=dm(I2, M2);
        dm(I4, M4)=AR;

        jump pzloop;


morethanthree: AR=dm(lastpeakfound);
        AR=AR+0;
        IF GT jump incptr;

        AR=1;
        dm(searchdownslope)=AR;

        AR=dm(notmaxpeak);
        AR=AR+0;
        IF GT jump wasnotmax;

        MX1=ay1;
        MY1=0x7332;
        MR=MX1*MY1(SS);
        AR=dm(lastpeakamp);
        ay0=MR1;
        dm(ninetenthsmax)=ay0;
        AR=AR-ay0;
        IF LT jump setlastpeak;

wasnotmax: AR=1;
        dm(lastpeakfound)=AR;

        AR=I6;
        AR=AR-2;
        I1=AR;

        AR=I7;
        AR=AR-1;
        I2=AR;

        M1=-1;
        M2=-1;
        L1=0;
        L2=0;

        I3=^threepeaks;
        I4=^surrzeros;
        M3=1;
        M4=1;

        AR=3;
        CNTR=AR;
        DO setthreepeaks2 UNTIL CE;
            AR=dm(I1, M1);
            dm(I3, M3)=AR;
            AR=dm(I2, M2);
setthreepeaks2:   dm(I4, M4)=AR;
        AR=dm(I2, M2);
        dm(I4, M4)=AR;

        jump incptr;

setlastpeak:  dm(lastpeakamp)=ay1;
        jump incptr;

not2:      AR=dm(slopepos1);
        AR=AR+0;
        IF LE jump not1;
        AR=ax1-ay1;
        IF EQ jump incptr;
        IF LT jump neg1;
        AR=1;
        dm(slopepos2)=AR;
        jump incptr;
```

```
neg1:      AR=0;
        dm(slopepos1)=AR;
        jump incptr;

not1:      AR=ax1-ay1;
        IF LE jump incptr;
        AR=1;
        dm(slopepos1)=AR;

incptr:    AR=dm(locptr);
        AR=AR+1;
        dm(locptr)=AR;
        ax0=ax1;
        AR=ax1-ay1;
        dm(prevslope)=AR;

pzloop:    nop;

afterpzloop:  AR=dm(lastpeakfound);
    AR=AR+0;
    IF GT jump endloop;

    ay0=5;
    ax0=dm(inputlabel);
    AR=ax0-ay0;
    IF GE jump doneitall;

    AR=1;
    dm(lastpeakfound)=AR;

    I1=I6;
    I2=I7;
    M1=-1;
    M2=-1;
    L1=0;
    L2=0;

    I3=^threepeaks;
    I4=^surrzeros;
    M3=1;
    M4=1;
    L3=0;
    L4=0;

    AR=dm(peakorzero);
    AR=AR+0;
    IF GT jump lookingforpeak;
    AR=dm(peakcount);
    AR=AR-1;

    IF GT jump enoughpeaks1;
        call badtap;
        jump endloop;
enoughpeaks1: AR=dm(I1, M1);
    AR=dm(I1, M1);
    AR=dm(I2, M2);
    AR=3;
    ay0=dm(peakcount);
    AR=AR-ay0;
    IF GT jump setcntrto3;
    AR=ay0;
    AR=AR-1;
    IF GT jump cntrfine16;
    call badtap;
    jump endloop;
cntrfine16: CNTR=AR;
    jump donesettingcntr1;
setcntrto3: CNTR=3;
donesettingcntr1:
        DO setthreepeaksend1 UNTIL CE;
        AR=dm(I1, M1);
        dm(I3, M3)=AR;
        AR=dm(I2, M2);
setthreepeaksend1: dm(I4, M4)=AR;
    AR=dm(I2, M2);
    dm(I4, M4)=AR;

    jump endloop;

lookingforpeak: AR=dm(peakcount);
    AR=AR+0;
    IF GT jump enoughpeaks2;
        call badtap;
        jump endloop;
enoughpeaks2: AR=dm(I1, M1);
    AR=dm(I2, M2);
    AR=3;
    ay0=dm(peakcount);
    AR=AR-ay0;
    IF GE jump setcntrto3a;
    AR=ay0;
    AR=AR+0;
    IF GT jump cntrfine17;
    AR=1;
cntrfine17: CNTR=ay0;

    jump donesettingcntr1a;
setcntrto3a:  CNTR=3;
donesettingcntr1a:
        DO set3peaksend UNTIL CE;
        AR=dm(I1, M1);
        dm(I3, M3)=AR;
        AR=dm(I2, M2);
set3peaksend: dm(I4, M4)=AR;
    AR=dm(I2, M2);
    dm(I4, M4)=AR;

endloop:   nop;
AR=^threepeaks;
call testthreepeaks;
doneitall: nop;
RTS;


transmitinput: {put input number in inputlabel}

    IMASK=0x0066;

    Uart0_Write(255);
    UART0_Write(105);
    AR=dm(inputlabel);
    call printdigit;
    Uart0_Write(255);

    UART0_Write(115);
    Uart0_Write(255);
    AR=dm(sum);
    call printhex;

    UART0_Write(109);
    Uart0_Write(255);

    AR=dm(inputlabel);
    ay0=5;
    AR=AR-ay0;
    IF LT jump usemaxpeak;
        AR=dm(max);
        jump donemaxpick;
usemaxpeak: AR=dm(maxpeak);
donemaxpick:  call printhex;

    UART0_Write(122);
    Uart0_Write(255);
    AR=dm(zerocrosscount);
    call printhex;

    IMASK=0x0006;
RTS;

clearpeakinfo:
    IMASK=0x0006;
    AR=%peakloc;
    CNTR=AR;
    AR=0;
    dm(peakcount)=AR;

    I5=^peakamphi;
    I6=^peakamplo;
    I7=^peakloc;
    L5=0;
    L6=0;
    L7=0;
    M5=1;
    M6=1;
    M7=1;
    AR=0;
    DO clearall UNTIL CE;
        DM(I5, M5)=AR;
        DM(I6, M6)=AR;
clearall:  DM(I7, M7)=AR;

    I5=^threepeaks;
    AR=3;
    CNTR=AR;
    AR=0;
    DO clearthreepeak UNTIL CE;
clearthreepeak: DM(I5, M5)=AR;


RTS;

clearzeroinfo:
    IMASK=0x0006;
    AR=%zerocrossloc;
    CNTR=AR;
    AR=0;
    dm(zerocrosscount)=AR;

    I5=^zerocrossloc;
    L5=0;
    M5=1;
    DO clearzeros UNTIL CE;
clearzeros: DM(I5, M5)=AR;

    I5=^surrzeros;
    AR=4;
    CNTR=AR;
    AR=0;
    DO clearsurrzeros UNTIL CE;
clearsurrzeros: DM(I5, M5)=AR;

RTS;

filter: {put the input in I5, input*save in I7}
    IMASK=0x0006;

    M5=1;
    M7=1;
    L5=0;
    L7=0;
    AR=dm(I7, M7);
    ay1=AR;
    AR=dm(I7, M7);
    ax0=AR;

    AR=dm(I5, M5);

    AR=dm(length);
    AR=AR-2;
    IF GT jump cntrfine18;
    AR=1;
cntrfine18: CNTR=AR;

    DO filterinput UNTIL CE;
        AR=dm(I7, M7);
        ax1=AR;

        ay0=0;
        AR=0;
        dm(filtersumhi)=AR;

        AR=ax1+ay1;
        ay0=AR;
        AR=ax0+ay0;
        ay0=AR;
        AR=0;
        AR=AR+C;
        dm(filtersumhi)=AR;
        AR=ax0+ay0;
        ay0=AR;
        AR=dm(filtersumhi);
        AR=AR+C;
        dm(filtersumhi)=AR;

        SR=LSHIFT AR BY -2(hi);
        AR=ay0;
        SR=SR OR LSHIFT AR BY -2(lo);
        dm(I5, M5)=SR0;

        ay1=ax0;
filterinput:  ax0=ax1;

RTS;

prepinput: {put the input in I5, label in AR, input*save in I7}
    IMASK=0x0006;
    dm(inputlabel)=AR;

    AR=I5;
    dm(saveinput)=AR;
    AR=I7;
    dm(inputsaveptr)=AR;

    call findinitialbias;

    AR=dm(saveinput);
    I5=AR;

    AR=dm(length);
    dm(maxminlength)=AR;

    AR=dm(inputlabel);
    ay0=5;
    AR=AR-ay0;
    IF NE jump notfive;
    AR=%mic;
    dm(maxminlength)=AR;
notfive:   call findmaxmin;

    AR=dm(saveinput);
    I5=AR;
    call normalize;

    AR=dm(max);
    SR=LSHIFT AR BY -4 (lo);
    AR=SR0;
    ay0=dm(maxsum);
    AR=AR+ay0;
    dm(maxsum)=AR;

    ay0=dm(max);
    AR=dm(largestmax);
    AR=AR-ay0;
    IF GE jump notlargestmax;
        dm(largestmax)=ay0;
notlargestmax: nop;

    AR=dm(saveinput);
    I5=AR;
    AR=dm(inputsaveptr);
    I7=AR;
    call filter;

    AR=dm(saveinput);
    I5=AR;
    call trimfront;
    call clearpeakinfo;
    call clearzeroinfo;
    AR=dm(saveinput);
    I5=AR;
    call findpeakzero;
    call transmitinput;
RTS;

savethreepeakinfo:
    {put ^threepeaks* in I5, ^surrzeros* in I7}
    IMASK=0x0006;
    M5=1;
    M7=1;
    L5=0;
    L7=0;

    I0=^threepeaks;
    I2=^surrzeros;
    M0=1;
    M2=1;
    L0=0;
    L2=0;

    CNTR=3;
    DO copy3peak UNTIL CE;
        AR=dm(I0, M0);
copy3peak: dm(I5, M5)=AR;

    CNTR=4;
    DO copyzeros UNTIL CE;
        AR=dm(I2, M2);
copyzeros: dm(I7, M7)=AR;

RTS;



findpeaks:
    IMASK=0x0006;
    call clearpeakinfo;
    I7=^xcorrhi;
    L7=0;
    M7=1;
    I4=^xcorrlo;
    L4=0;
    M4=1;

    I5=^peakamphi;
    L5=0;
    M5=1;
    I6=^peakamplo;
    L6=0;
    M6=1;
    L1=0;
    M1=1;
    I0=^peakloc;
    L0=0;
    M0=1;
    AR=0;
    dm(peakcount)=AR;
    dm(slopepos1)=AR;
    dm(slopepos2)=AR;

    AR=1;
    dm(locptr)=AR;
    AR=dm(corrlength);
    AR=AR+0;
    IF GT jump cntrokay7;
        call badtap;
        jump pfend;
cntrokay7: CNTR=AR;
    MR1=dm(I7, M7);
    MR0=dm(I4, M4);

    DO peakloop UNTIL CE;

        SR1=dm(I7, M7);
        SR0=dm(I4, M4);

        AR=dm(slopepos2);
        AR=AR+0;
        IF LE jump nots2;
        ay0=MR0;
        ax0=SR0;
        AR=ax0-ay0;
        ay0=SR1;
        ax0=MR1;
        AR=ay0-ax0+C-1;
        IF GE jump ploop1;

        AR=0;
        dm(slopepos2)=AR;
        dm(slopepos1)=AR;

        AR=%peakloc;
        ay0=AR;
        AR=dm(peakcount);
        AR=AR-ay0;
        if LE jump continue;
        pop CNTR;
        pop PC;
        pop LOOP;
        jump pfend;

continue:  AR=dm(peakcount);
        AR=AR+1;
        dm(peakcount)=AR;

        ay0=dm(locptr);
        dm(I0, M0)=ay0;
        AR=^xcorrhi;
        AR=AR+ay0;
        AR=AR-1;
        I1=AR;
        AR=dm(I1, M1);
        dm(I5, M5)=AR;

        AR=^xcorrlo;
        AR=AR+ay0;
        AR=AR-1;
        I1=AR;
        AR=dm(I1, M1);
        dm(I6, M6)=AR;
        jump ploop1;

nots2:    AR=dm(slopepos1);
```

```
        AR=AR+0;
        IF LE jump nots1;
        ay0=MR0;
        ax0=SR0;
        AR=ax0-ay0;
        ay0=SR1;
        ax0=MR1;
        AR=ay0-ax0+C-1;

        IF LT jump negs1;
        AR=1;
        dm(slopepos2)=AR;
        jump ploop1;
negs1:    AR=0;
        dm(slopepos1)=AR;
        jump ploop1;

nots1:    ay0=MR0;
        ax0=SR0;
        AR=ax0-ay0;
        ay0=SR1;
        ax0=MR1;
        AR=ay0-ax0+C-1;
        IF LT jump ploop1;
        AR=1;
        dm(slopepos1)=AR;

ploop1:   AR=dm(locptr);
        AR=AR+1;
        dm(locptr)=AR;

        MR0=SR0;
peakloop:  MR1=SR1;

pfend: AR=0;
    dm(locptr)=AR;
RTS;

findcorrdiff:
    IMASK=0x0006;
    I5=^peakamphi;
    I6=^peakamplo;
    I7=^peakloc;
    M5=1;
    M6=1;
    M7=1;
    L5=0;
    L6=0;
    L7=0;

    AR=0x8000;
    dm(maxcorrpeakhi)=AR;
    dm(secondcorrpeakhi)=AR;
    dm(maxcorrpeaklo)=AR;
    dm(secondcorrpeaklo)=AR;

    AR=0;
    dm(maxcorrpeakloc)=AR;
    dm(secondcorrpeakloc)=AR;

    AR=dm(peakcount);
    AR=AR+0;
    IF GT jump cntrokay8;
        call badtap;
        jump donefindcorrdiff;
cntrokay8: CNTR=AR;
    DO findmaxpeak UNTIL CE;
        ax0=dm(maxcorrpeaklo);
        ay0=dm(I6, M6);
        AR=ax0-ay0;
        ay1=dm(maxcorrpeakhi);
        ax1=dm(I5, M5);
        AR=ay1-ax1+C-1;
        IF GE jump notnewcorrmax;
            AR=dm(maxcorrpeaklo);
            dm(secondcorrpeaklo)=AR;
            AR=dm(maxcorrpeakhi);
            dm(secondcorrpeakhi)=AR;
            AR=dm(maxcorrpeakloc);
            dm(secondcorrpeakloc)=AR;

            dm(maxcorrpeaklo)=ay0;
            dm(maxcorrpeakhi)=ax1;
            AR=dm(I7, M7);
            dm(maxcorrpeakloc)=AR;
            jump findmaxpeak;
notnewcorrmax: ax0=dm(secondcorrpeaklo);
        AR=ax0-ay0;
        ay1=dm(secondcorrpeakhi);
        aR=ay1-ax1+C-1;
        IF GE jump notsecondpeak;
            dm(secondcorrpeaklo)=ay0;
            dm(secondcorrpeakhi)=ax1;
            AR=dm(I7, M7);
            dm(secondcorrpeakloc)=AR;
            jump findmaxpeak;

notsecondpeak: AR=dm(I7, M7);
findmaxpeak:  nop;

    AR=dm(corrlength);
    AR=AR+1;
    SR=LSHIFT AR BY -1 (lo);
    ax0=SR0;

    ay0=dm(maxcorrpeakloc);
    AR=ax0-ay0;
    dm(maxcorrdiff)=AR;

    ay0=dm(secondcorrpeakloc);
    AR=ax0-ay0;
    dm(secondcorrdiff)=AR;

    ax0=dm(inp1startfinal);
    ay0=dm(inp2startfinal);
    AR=ax0-ay0;
    ax1=AR;
    ay0=dm(maxcorrdiff);
    AR=ax1+ay0;
    dm(maxcorrdiff)=AR;
    ay0=dm(secondcorrdiff);
    AR=ax1+ay0;
    dm(secondcorrdiff)=AR;
donefindcorrdiff: nop;
RTS;

transmitknuckle:

    IMASK=0x0066;

    UART0_Write(120);
    Uart0_Write(255);
    AR=dm(chisqminhi);
    call printhex;
    AR=dm(chisqminlo);
    call printhex;

    UART0_Write(99);
    Uart0_Write(255);
    AR=dm(maxcorrdiff);
    call printhex;
    AR=dm(secondcorrdiff);
    call printhex;

    UART0_Write(97);
    Uart0_Write(255);
    AR=dm(maxcorrpeakhi);
    call printhex;
    AR=dm(maxcorrpeaklo);
    call printhex;
    Uart0_Write(255);
    AR=dm(secondcorrpeakhi);
    call printhex;
    AR=dm(secondcorrpeaklo);
    call printhex;

    IMASK=0x0006;
RTS;


findmetaltapstart: {put input in I5}
    IMASK=0x0006;

    AR=I5;
    dm(saveinput)=AR;
    M5=1;
    L5=0;

    AR=70;
    dm(maxminlength)=AR;
    call findmaxmin;

    AR=dm(max);
    dm(threshhold)=AR;

    SR=LSHIFT AR BY -1 (lo);
    ay0=SR0;

    AR=20;
    ay1=AR;

    AR=dm(threshhold);
    AR=AR+ay0;
    AR=AR+ay1;
    dm(threshhold)=AR;

    AR=dm(saveinput);
    I5=AR;

    AR=1;
    dm(locptr)=AR;

    AR=1;
    dm(metaltapstart)=AR;

    ay0=dm(threshhold);

    AR=dm(length);
    CNTR=AR;
    DO newmetalalg UNTIL CE;
        AR=dm(I5, M5);
        AR=AR-ay0;
        IF LE jump notreachedstart;
        AR=dm(locptr);
        dm(metaltapstart)=AR;
        POP LOOP;
        POP CNTR;
        POP PC;
        RTS;
notreachedstart:
    AR=dm(locptr);
    AR=AR+1;
newmetalalg:  dm(locptr)=AR;
donemetalthres: nop;

RTS;

badtap:
    AR=1;
    dm(wasbadtap)=AR;
RTS;

PWMSYNC_ISR:
    {IMASK=0x0066;
    uart0_write(115);
    uart0_write(121);
    uart0_write(110);
    uart0_write(99);
    uart0_write(105);
    uart0_write(110);
    uart0_write(32);}

    IMASK=0x0006;

    AR=dm(startdata);
    AR=AR+0;
    if EQ jump notinited;

    AR=dm(bangcount);
    AR=AR+0;
    IF EQ jump startprocessing;
    IF LE jump delayzero;
    AR=AR-1;
    dm(bangcount)=AR;
    ADC_Read(ADC0, Offset_0to3);
    DM(I0, M0)=AR;
    jump next2;

delayzero: AR=dm(trigger);
    AR=AR+0;
    if LE jump next1;

    AR=dm(aftertriggerlength);
    ay0=AR;
    AR=dm(trigger);
    AR=AR+1;
    dm(trigger)=AR;
    AR=AR-ay0;
    if LT jump next1;

startprocessing:  AR=dm(bangcount);
    AR=AR+0;
    IF EQ jump donebang;

    AR=600;
    dm(bangcount)=AR;
    jump next1;

donebang:  call copyinputs;

    AR=0;
    dm(zerocrosssum)=AR;

    AR=0;
    dm(maxsum)=AR;

    AR=0;
    dm(largestmax)=AR;

    AR=0x0001;
    I5=^input1;
    I7=^input1save;
    call prepinput;
    I5=^threepeaks1;
    I7=^surrzeros1;
    call savethreepeakinfo;
    AR=^threepeaks1;
    call testthreepeaks;
    AR=dm(zerocrosscount);
    dm(zerocrosssum)=AR;

    AR=0x0002;
    I5=^input2;
    I7=^input2save;
    call prepinput;
    I5=^threepeaks2;
    I7=^surrzeros2;
    call savethreepeakinfo;
    AR=^threepeaks2;
    call testthreepeaks;
    AR=dm(zerocrosssum);
    ay0=dm(zerocrosscount);
    AR=AR+ay0;
    dm(zerocrosssum)=AR;

    AR=0x0003;
    I5=^input3;
    I7=^input3save;
    call prepinput;
    I5=^threepeaks3;
    I7=^surrzeros3;
    call savethreepeakinfo;
    AR=^threepeaks3;
    call testthreepeaks;
    AR=dm(zerocrosssum);
    ay0=dm(zerocrosscount);
    AR=AR+ay0;
    dm(zerocrosssum)=AR;

    AR=0x0004;
    I5=^input4;
    I7=^input4save;
    call prepinput;
    I5=^threepeaks4;
    I7=^surrzeros4;
    call savethreepeakinfo;
    AR=^threepeaks4;
    call testthreepeaks;
    AR=dm(zerocrosssum);
    ay0=dm(zerocrosscount);
    AR=AR+ay0;
    dm(zerocrosssum)=AR;

    AR=0x0005;
    I5=^mic;
    I7=^micsave;
    call prepinput;

    AR=0x0006;
    I5=^mic2;
    I7=^mic2save;
    call prepinput;

    IMASK=0x0066;

    AR=dm(metaltapthres);
    ay0=dm(zerocrosssum);
    AR=AR-ay0;
    IF LE jump metaltap;

knuckletap: UART0_Write(75);
        Uart0_Write(255);

    IMASK=0x0006;

    AR=^threepeaks1;
    dm(inp1peaksptr)=AR;
    AR=^surrzeros1;
    dm(inp1surrzerosptr)=AR;
    AR=^threepeaks3;
    dm(inp2peaksptr)=AR;
    AR=^surrzeros3;
    dm(inp2surrzerosptr)=AR;

    call choosepeaks;

    AR=^input1;
    dm(inp1ptr)=AR;
    AR=^input3;
    dm(inp2ptr)=AR;

    call findchisqmin;
    call correlate;
    call findpeaks;
    call findcorrdiff;
    IMASK=0x0066;
    UART0_Write(49);
    UART0_Write(51);
    Uart0_Write(255);
    call transmitknuckle;

    IMASK=0x0006;
    call save13info;

    AR=^threepeaks2;
    dm(inp1peaksptr)=AR;
    AR=^surrzeros2;
    dm(inp1surrzerosptr)=AR;
    AR=^threepeaks4;
    dm(inp2peaksptr)=AR;
    AR=^surrzeros4;
    dm(inp2surrzerosptr)=AR;

    call choosepeaks;

    AR=^input2;
    dm(inp1ptr)=AR;
    AR=^input4;
    dm(inp2ptr)=AR;

    call findchisqmin;
    call copycorrinputs;
    call correlate;
    call findpeaks;
    call findcorrdiff;
    IMASK=0x0066;
    UART0_Write(50);
    UART0_Write(52);
    Uart0_Write(255);
    call transmitknuckle;

metaltap:  IMASK=0x0066;

    UART0_Write(77);
    Uart0_Write(255);

    I5=^input1save;
    call findmetaltapstart;
    AR=dm(metaltapstart);
    call printhex;

    I5=^input2save;
    call findmetaltapstart;
    AR=dm(metaltapstart);
    call printhex;

    I5=^input3save;
    call findmetaltapstart;
    AR=dm(metaltapstart);
    call printhex;

    I5=^input4save;
    call findmetaltapstart;
    AR=dm(metaltapstart);
    call printhex;

nexttap:   nop;

    IMASK=0x0066;

    AR=dm(wasbadtap);
    call printdigit;

    Uart0_Write(255);
    Uart0_Write(255);
    Uart0_Write(255);


    IMASK=0x0006;

    call initvalues;

    AR=dm(largestmax);
    SR=LSHIFT AR BY 4 (lo);
    ay0=SR0;
    AR=0x0050;
    AR=AR+ay0;
    dm(decaythres)=AR;

next1:
    IMASK=0x0006;

    ADC_Read(ADC4, Offset_4to7);
    dm(I1, M1)=AR;
    dm(inp1value)=AR;

    ADC_Read(ADC5, Offset_4to7);
    dm(I2, M2)=AR;
    dm(inp2value)=AR;

    ADC_Read(ADC6, Offset_4to7);
    dm(I3, M3)=AR;
    dm(inp3value)=AR;

    ADC_Read(ADC7, Offset_4to7);
    dm(I4, M4)=AR;
    dm(inp4value)=AR;

    ADC_Read(ADC0, Offset_0to3);
    dm(inp5value)=AR;
    DM(I0, M0)=AR;

    ADC_Read(ADC1, Offset_0to3);
    dm(inp6value)=AR;
    DM(I5, M5)=AR;

    AR=dm(startsamplecount);
    AR=AR+0;
    IF LE jump okaytriggeralready;
        AR=AR-1;
        dm(startsamplecount)=AR;
        jump next2;
okaytriggeralready:

    AR=dm(trigger);
    AR=AR+0;
    if NE jump next2;

    AR=dm(inp1value);
    AR=AR+0;
    if GT JUMP nn1;
        AR=-AR;
nn1:   SR= LSHIFT AR by 0 (LO);
    AR=dm(inp2value);
    AR=AR+0;
    if GT JUMP nn2;
        AR=-AR;
nn2:   SR=SR or LSHIFT AR by 0 (LO);
    AR=dm(inp3value);
    AR=AR+0;
    if GT JUMP nn3;
        AR=-AR;
nn3:   SR=SR or LSHIFT AR by 0 (LO);
    AR=dm(inp4value);
    AR=AR+0;
    if GT JUMP nn4;
        AR=-AR;
nn4:   SR=SR or LSHIFT AR by 0 (LO);
    Ax0=SR0;
    ay0=dm(tapthres);
    AR=dm(decaythres);
    AR=AR-ay0;
    IF LE jump lessthanthres;
    ay0=dm(decaythres);
    ax1=ay0;
    ay1=dm(decaymult);
    AR=ax1-ay1;
    IF GE jump decaythrespos;
    AR=0;
decaythrespos: dm(decaythres)=AR;

lessthanthres: AR=Ax0-ay0;
    IF LT JUMP next2;
    jump starttrigger;

    AR=dm(inp5value);
    ay0=dm(bangthres);
    AR=AR-ay0;
    if GT jump starttrigger;

    jump next2;

starttrigger:  AR=1;
    dm(trigger)=AR;
next2: nop;
notinited: nop;

    IFC=0x80;
    IMASK=0x0206;
RTI;


PWMTRIP_ISR:
    IMASK=0x0006;
    AR=dm(pwmtripcount);
    AR=AR+1;
    dm(pwmtripcount)=AR;

    write_dm(PWMTM,0x104);
    write_dm(PWMCHA,0);
    write_dm(PWMCHB,0);
    write_dm(PWMCHC,0);
    IMASK=0x0206;
rti;


.ENDMOD;
```

chisq.dsp

```
{****** chisq.dsp ************************}
{****** Library for doing chi-squared ***}
{****** and cross-correlation method *****}



.MODULE/RAM/SEG=USER_PM2 CHISQ;

.ENTRY mult32by16;
.ENTRY divide;
.ENTRY copychisqinputs;
.ENTRY findmulfact;
.ENTRY chisqnormalize;
.ENTRY findchisqmin;
.ENTRY comparechisqmin;
.ENTRY chisqfunc;
.ENTRY choosepeaks;
.ENTRY copycorrinputs;
.ENTRY correlate;

.EXTERNAL badtap;
.EXTERNAL findmaxmin;
.EXTERNAL input1;
.EXTERNAL input2;
.EXTERNAL tempinput;
.EXTERNAL locptr;
.EXTERNAL max;
.EXTERNAL threshhold;
.EXTERNAL min;
.EXTERNAL trimstart;

.EXTERNAL inp1peaksptr;
.EXTERNAL inp2peaksptr;
.EXTERNAL inp1surrzerosptr;
.EXTERNAL inp2surrzerosptr;
.EXTERNAL inp1start;
.EXTERNAL inp1end;
.EXTERNAL inp2start;
.EXTERNAL inp2end;
.EXTERNAL inp1start2;
.EXTERNAL inp2start2;
.EXTERNAL inp1end2;
.EXTERNAL inp2end2;
.EXTERNAL inp1num;
.EXTERNAL inp2num;
.EXTERNAL inp1startfinal;
.EXTERNAL inp1endfinal;
.EXTERNAL inp2startfinal;
.EXTERNAL inp2endfinal;
.EXTERNAL finalchisqminlo;
.EXTERNAL finalchisqminhi;
.EXTERNAL inp1ptr;
.EXTERNAL inp2ptr;
.EXTERNAL chisqinput1;
.EXTERNAL chisqinput2;
.EXTERNAL chisqlen1;
.EXTERNAL chisqlen2;
.EXTERNAL chisqmaxlen;
.EXTERNAL chisqminlo;
.EXTERNAL chisqminhi;
.EXTERNAL checkdist;
.EXTERNAL newmin;
.EXTERNAL shiftinput;
.EXTERNAL mulfacthi;
.EXTERNAL mulfactlo;
.EXTERNAL maxminlength;
.EXTERNAL corrlength;

.EXTERNAL xcorrhi;
.EXTERNAL xcorrlo;

mult32by16:
    {32-bit in MX1, MX0, }
    {16-bit in MY1, result in SR0}

    MR=MX0*MY1(US);
    MY0=MR0;
    MR0=MR1;
    AR=0x0000;
    MR1=AR;
    MR=MR+MX1*MY1(SS);
    SR=LSHIFT MR0 BY -18 (hi);
    AR=MY0;
    SR=SR OR LSHIFT AR BY -18 (lo);
RTS;


divide:
    {MSW of dividend in AY1,}
    {LSW in AY0, divisor in AX0; result in SR0}
    DIVS AY1, AX0;
    DIVQ AX0; DIVQ AX0; DIVQ AX0; DIVQ AX0; DIVQ AX0;
    DIVQ AX0; DIVQ AX0; DIVQ AX0; DIVQ AX0; DIVQ AX0;
    DIVQ AX0; DIVQ AX0; DIVQ AX0; DIVQ AX0; DIVQ AX0;
    AR=ay0;
    SR=LSHIFT AR BY 1 (LO);
RTS;

copychisqinputs:
    {put inp*start* in ay0,}
    {ay1, lengths in chisqlen1,2,}
    {addresses of inputs in inp1ptr, inp2ptr}

    AR=dm(chisqlen1);
    AR=AR+0;
    IF GT jump cntrokay9;
        call badtap;
        jump donecopychisqinputs;
cntrokay9: CNTR=AR;
    I1=^chisqinput1;
    M1=1;
    AR=dm(inp1ptr);
    AR=AR-1;
    AR=AR+ay0;
    I2=AR;
    M2=1;
    DO copyinp1 UNTIL CE;
        AR=dm(I2, M2);
copyinp1:  dm(I1, M1)=AR;

    AR=dm(chisqlen2);
    AR=AR+0;
    IF GT jump cntrokay10;
        call badtap;
        jump donecopychisqinputs;
cntrokay10: CNTR=AR;
    I1=^chisqinput2;
    AR=dm(inp2ptr);
    AR=AR-1;
    AR=AR+ay1;
    I2=AR;
    DO copyinp2 UNTIL CE;
        AR=dm(I2, M2);
copyinp2:  dm(I1, M1)=AR;
donecopychisqinputs:  nop;
RTS;

findmulfact:
    ay1=0x07FF;
    ay0=0xF000;

    AF=pass 0xF000;
    MR0=dm(max);
    AR=MR0 AND AF;
    IF EQ jump notbyte1;
    ax0=dm(max);
    call divide;
    AR=0x0000;
    dm(mulfacthi)=AR;
    dm(mulfactlo)=SR0;
    jump donefact;

notbyte1:  AR=0x0F00;
    AF=pass AR;
    AR=MR0 AND AF;
    IF EQ jump notbyte2;
    SR=LSHIFT MR0 BY 3 (lo);
    ax0=SR0;
    call divide;
    AR=SR0;
    SR=LSHIFT AR BY 3 (lo);
    dm(mulfacthi)=SR1;
    dm(mulfactlo)=SR0;
    jump donefact;

notbyte2:
    AR=0x00F0;
    AF=pass AR;
    AR=MR0 AND AF;
    IF EQ jump notbyte3;
    SR=LSHIFT MR0 BY 7 (lo);
    ax0=SR0;
    call divide;
    AR=SR0;
    SR=LSHIFT AR BY 7 (lo);
    dm(mulfacthi)=SR1;
    dm(mulfactlo)=SR0;
    jump donefact;

notbyte3:
    SR=LSHIFT MR0 BY 11 (lo);
    ax0=SR0;
    call divide;
    AR=SR0;
    SR=LSHIFT AR BY 11 (lo);
    dm(mulfacthi)=SR1;
    dm(mulfactlo)=SR0;

donefact:  nop;
RTS;

chisqnormalize:
    I5=^chisqinput1;
    AR=dm(chisqlen1);
    dm(maxminlength)=AR;
    call findmaxmin;
    call findmulfact;

    I5=^chisqinput1;
    M5=1;
    L5=0;
    I6=I5;
    M6=1;
    L6=0;
    AR=dm(chisqlen1);
    AR=AR+0;
    IF GT jump cntrokay11;
        call badtap;
        jump donechisqnormalize;
cntrokay11: CNTR=AR;
    MX1=dm(mulfacthi);
    MX0=dm(mulfactlo);
    DO multbyfact1 UNTIL CE;
        MY1=dm(I5, M5);
        call mult32by16;
multbyfact1:  dm(I6, M6)=SR0;

    I5=^chisqinput2;
    AR=dm(chisqlen2);
    dm(maxminlength)=AR;
    call findmaxmin;
    call findmulfact;

    I5=^chisqinput2;
    I6=I5;
    AR=dm(chisqlen2);
    AR=AR+0;
    IF GT jump cntrokay12;
        call badtap;
        jump donechisqnormalize;
cntrokay12: CNTR=AR;
    MX1=dm(mulfacthi);
    MX0=dm(mulfactlo);
    DO multbymax1 UNTIL CE;
        MY1=dm(I5, M5);
        call mult32by16;
multbymax1: dm(I6, M6)=SR0;
donechisqnormalize: nop;
RTS;

findchisqmin:

    AR=0x7FFF;
    dm(chisqminhi)=AR;
    dm(chisqminlo)=AR;

    L1=0;
    L2=0;
    AR=dm(inp1num);
    AR=AR+0;

    IF GT jump notboth1;
    AR=dm(inp2num);
    AR=AR+0;
    IF GT jump notboth1;
    AR=dm(inp1start);
    dm(inp1startfinal)=AR;
    AR=dm(inp1end);
    dm(inp1endfinal)=AR;
    AR=dm(inp2start);
    dm(inp2startfinal)=AR;
    AR=dm(inp2end);
    dm(inp2endfinal)=AR;

    ay0=dm(inp1start);
    AR=dm(inp1end);
    AR=AR-ay0;
    AR=AR+1;
    dm(chisqlen1)=AR;

    ay1=dm(inp2start);
    AR=dm(inp2end);
    AR=AR-ay1;
    AR=AR+1;
    dm(chisqlen2)=AR;

    call copychisqinputs;
    call chisqnormalize;
    call chisqfunc;
    AR=dm(chisqminhi);
    dm(finalchisqminhi)=AR;
    AR=dm(chisqminlo);
    dm(finalchisqminlo)=AR;
    jump donechisqmin;

notboth1:
    AR=dm(inp1start);
    ay0=AR;
    AR=dm(inp1end);
    AR=AR-ay0;
    AR=AR+1;
    dm(chisqlen1)=AR;

    ay1=dm(inp2start);
    AR=dm(inp2end);
    AR=AR-ay1;
    AR=AR+1;
    dm(chisqlen2)=AR;

    call copychisqinputs;
    call chisqnormalize;
    call chisqfunc;
    AR=dm(chisqminhi);
    dm(finalchisqminhi)=AR;
    AR=dm(chisqminlo);
    dm(finalchisqminlo)=AR;

    AR=dm(inp1start);
    dm(inp1startfinal)=AR;
    AR=dm(inp1end);
    dm(inp1endfinal)=AR;
    AR=dm(inp2start);
    dm(inp2startfinal)=AR;
    AR=dm(inp2end);
    dm(inp2endfinal)=AR;

    AR=dm(inp1num);
    AR=AR+0;
    IF GT jump twoinp1;

twoinp2:
    AR=dm(inp1start);
    ay0=AR;
    AR=dm(inp1end);
    AR=AR-ay0;
    AR=AR+1;
    dm(chisqlen1)=AR;

    AR=dm(inp2start2);
    ay1=AR;
    AR=dm(inp2end2);
    AR=AR-ay1;
    AR=AR+1;
    dm(chisqlen2)=AR;

    call comparechisqmin;
    AR=dm(newmin);
    AR=AR+0;
    IF EQ jump oldmin1;
    AR=dm(inp1start);
    dm(inp1startfinal)=AR;
    AR=dm(inp1end);
    dm(inp1endfinal)=AR;
    AR=dm(inp2start2);
    dm(inp2startfinal)=AR;
    AR=dm(inp2end2);
    dm(inp2endfinal)=AR;
oldmin1:   jump donechisqmin;

twoinp1:
    AR=dm(inp2num);
    AR=AR+0;
    IF GT jump twoboth;

    ay0=dm(inp1start2);
    AR=dm(inp1end2);
    AR=AR-ay0;
    AR=AR+1;
    dm(chisqlen1)=AR;

    ay1=dm(inp2start);
    AR=dm(inp2end);
    AR=AR-ay1;
    AR=AR+1;
    dm(chisqlen2)=AR;

    call comparechisqmin;
    AR=dm(newmin);
    AR=AR+0;
    IF EQ jump oldmin2;
    AR=dm(inp1start2);
    dm(inp1startfinal)=AR;
    AR=dm(inp1end2);
    dm(inp1endfinal)=AR;
    AR=dm(inp2start);
    dm(inp2startfinal)=AR;
    AR=dm(inp2end);
    dm(inp2endfinal)=AR;

oldmin2:   jump donechisqmin;

twoboth:
    AR=dm(inp1start2);
    ay0=AR;
    AR=dm(inp1end2);
    AR=AR-ay0;
    AR=AR+1;
    dm(chisqlen1)=AR;

    ay1=dm(inp2start);
    AR=dm(inp2end);
    AR=AR-ay1;
    AR=AR+1;
    dm(chisqlen2)=AR;

    call comparechisqmin;
    AR=dm(newmin);
    AR=AR+0;
    IF EQ jump oldmin3;
    AR=dm(inp1start2);
    dm(inp1startfinal)=AR;
    AR=dm(inp1end2);
    dm(inp1endfinal)=AR;
    AR=dm(inp2start);
    dm(inp2startfinal)=AR;
    AR=dm(inp2end);
    dm(inp2endfinal)=AR;

oldmin3:
    AR=dm(inp1start);
    ay0=AR;
    AR=dm(inp1end);
    AR=AR-ay0;
    AR=AR+1;
    dm(chisqlen1)=AR;

    AR=dm(inp2start2);
    ay1=AR;
    AR=dm(inp2end2);
    AR=AR-ay1;
    AR=AR+1;
    dm(chisqlen2)=AR;

    call comparechisqmin;
    AR=dm(newmin);
    AR=AR+0;
    IF EQ jump oldmin4;
    AR=dm(inp1start);
    dm(inp1startfinal)=AR;
    AR=dm(inp1end);
    dm(inp1endfinal)=AR;
    AR=dm(inp2start2);
    dm(inp2startfinal)=AR;
    AR=dm(inp2end2);
    dm(inp2endfinal)=AR;

oldmin4:
    AR=dm(inp1start2);
    ay0=AR;
    AR=dm(inp1end2);
    AR=AR-ay0;
    AR=AR+1;
    dm(chisqlen1)=AR;

    AR=dm(inp2start2);
    ay1=AR;
    AR=dm(inp2end2);
    AR=AR-ay1;
    AR=AR+1;
    dm(chisqlen2)=AR;

    call comparechisqmin;
    AR=dm(newmin);
    AR=AR+0;
    IF EQ jump donechisqmin;
    AR=dm(inp1start2);
    dm(inp1startfinal)=AR;
    AR=dm(inp1end2);
    dm(inp1endfinal)=AR;
    AR=dm(inp2start2);
    dm(inp2startfinal)=AR;
    AR=dm(inp2end2);
    dm(inp2endfinal)=AR;

donechisqmin: nop;

RTS;


comparechisqmin:
    AR=0;
    dm(newmin)=AR;

    call copychisqinputs;
    call chisqnormalize;
    call chisqfunc;
    ay0=dm(finalchisqminlo);
    AR=dm(chisqminlo);
    AR=AR-ay0;
    ax0=dm(finalchisqminhi);
    ay0=dm(chisqminhi);
    AR=ay0-ax0+C-1;
    IF GE jump notmin1;
    AR=dm(chisqminlo);
    dm(finalchisqminlo)=AR;
    AR=dm(chisqminhi);
    dm(finalchisqminhi)=AR;
    AR=1;
    dm(newmin)=AR;
notmin1:   nop;
RTS;


chisqfunc: {put inputs in chisqinput1,2, lengths in chisqlen1,2}

    I1=^chisqinput1;
    AR=dm(chisqlen1);
    ay0=AR;
    AR=I1;
    AR=AR+ay0;
    AR=AR-1;
    I1=AR;
    M1=-1;

    AR=dm(checkdist);
    ay1=AR;
    AR=I1;
    AR=AR+ay1;
    I3=AR;
    M3=-1;
    AR=AR+1;
    I5=AR;
    M5=1;
    I2=^chisqinput2;
    AR=dm(chisqlen2);
    ax0=AR;

    AR=ax0-ay0;
    IF GT jump secondlonger;
    AR=ay0;
    dm(chisqmaxlen)=AR;
    jump donemaxlen;
secondlonger:  nop;
    AR=ax0;
    dm(chisqmaxlen)=AR;

donemaxlen: AR=I2;
    ay1=ax0;
    AR=AR+ay1;
    AR=AR-1;
    I2=AR;
    M2=-1;

    AR=dm(checkdist);
    ay0=AR;
    AR=AR+ay0;
    ay0=AR;
    AR=I2;
    AR=AR+ay0;
    I4=AR;
    M4=-1;
    AR=AR+1;
    I6=AR;
    M6=1;

    AR=dm(chisqlen1);
    AR=AR+0;
    IF GT jump cntrokay13;
        call badtap;
        jump donechisqfunc;
cntrokay13: CNTR=AR;
    DO shiftinput1 UNTIL CE;
        AR=dm(I1, M1);
shiftinput1:  dm(I3, M3)=AR;

    AR=dm(chisqlen2);
    AR=AR+0;
    IF GT jump cntrokay14;
        call badtap;
        jump donechisqfunc;
cntrokay14: CNTR=AR;
    DO shiftinput2 UNTIL CE;
        AR=dm(I2, M2);
shiftinput2:  dm(I4, M4)=AR;

    AR=dm(checkdist);
    AR=AR+0;
    IF GT jump cntrfine23;
    CNTR=1;
cntrfine23: CNTR=AR;
    AR=0;
    DO addszeros1 UNTIL CE;
addszeros1: dm(I3, M3)=AR;

    AR=dm(checkdist);
    ay0=AR;
    AR=AR+ay0;
    IF GT jump cntrfine24;
    CNTR=1;
cntrfine24: CNTR=AR;
    AR=0;
    DO addszeros2 UNTIL CE;
addszeros2: dm(I4, M4)=AR;

    AR=dm(chisqlen2);
    ay0=AR;
    AR=dm(checkdist);
    AR=AR+ay0;
    IF GT jump cntrfine25;
    CNTR=1;
cntrfine25: CNTR=AR;
    AR=0;
    DO addezeros1 UNTIL CE;
addezeros1: dm(I5, M5)=AR;

    AR=dm(chisqlen1);
    ay0=AR;
    AR=dm(checkdist);
    AR=AR+ay0;
    IF GT jump cntrfine26;
    CNTR=1;
cntrfine26: CNTR=AR;
    AR=0;
    DO addezeros2 UNTIL CE;
addezeros2: dm(I6, M6)=AR;

    I1=^chisqinput1;
    AR=^chisqinput2;
    ax0=AR;
    AR=dm(checkdist);
    ay0=AR;
    AR=AR+ay0;
    ay0=AR;
    AR=ax0+ay0;
    I2=AR;
    M1=1;
    M2=-1;
    L1=0;
    L2=0;

    M3=1;
    M4=1;
    L3=0;
    L4=0;

    AR=0x7FFF;
    dm(chisqminhi)=AR;
    dm(chisqminlo)=AR;

    AR=dm(checkdist);
    ay0=AR;
    AR=AR+ay0;
    IF GT jump cntrfine27;
    CNTR=1;
cntrfine27: CNTR=AR;

    DO chisqloop UNTIL CE;
    MR0=0;
    MR1=0;
    MR2=0;
    I3=I1;
    I4=I2;
    AR=dm(chisqmaxlen);
    ax0=AR;
    AR=dm(checkdist);
    ay0=AR;
    AR=AR+ay0;
    ay0=AR;
    AR=ax0+ay0;
    AR=AR+0;
    IF GT jump cntrfine28;
    CNTR=1;
cntrfine28: CNTR=AR;

    DO dataloop UNTIL CE;
        ax0=dm(I3, M3);
        ay0=dm(I4, M4);
        AR=ax0-ay0;
        MY0=AR;
        MX0=AR;
dataloop:  MR=MR+MX0*MY0(SS);

    SR=LSHIFT MR2 BY 11 (HI);
    SR=SR OR LSHIFT MR1 BY -5 (HI);
    SR=SR OR LSHIFT MR0 BY -5 (LO);
    AR=dm(chisqminlo);
    ay0=SR0;
    AR=AR-ay0;
    ay0=dm(chisqminhi);
    AR=SR1;
    AR=ay0-AR+C-1;
    IF LE jump notmin;
    AR=SR0;
    dm(chisqminlo)=AR;
    AR=SR1;
    dm(chisqminhi)=AR;
notmin: AR=dm(I2, M2);

    MR1=0;
chisqloop: MR0=0;
donechisqfunc: nop;
RTS;

choosepeaks:
    {put addresses of both input}
    {threepeaks, surrzeros in}
    {inp1peaksptr, inp1surrzerosptr,}
    {inp2peaksptr, inp2surrzerosptr}

    M1=1;
    M3=1;
    M4=1;
    M6=1;
    L1=0;
    L3=0;
    L4=0;
    L6=0;
    AR=0;
    dm(inp1num)=AR;
    dm(inp2num)=AR;

    AR=dm(inp1peaksptr);
    I1=AR;
    AR=dm(inp2peaksptr);
    I4=AR;
    AR=dm(I1, M1);
    AR=AR+0;
    IF GT jump okay1;
        call badtap;
        jump donepick;
okay1: AR=dm(I4, M4);
    AR=AR+0;
    IF GT jump okay2;
        call badtap;
        jump donepick;
okay2: M1=0;
    M4=0;
    AR=dm(I1, M1);
    AR=AR+0;
    IF GT jump p21fine;

    AR=dm(I4, M4);
    AR=AR+0;
    IF GT jump p22fine;
    AR=dm(inp1surrzerosptr);
    I3=AR;
    M3=1;
    AR=dm(I3, M3);
    dm(inp1end)=AR;
    AR=AR+0;
    IF GT jump okay3;
        call badtap;
        jump donepick;
okay3: AR=dm(I3, M3);
    dm(inp1start)=AR;
    AR=AR+0;
    IF GT jump okay4;
        call badtap;
        jump donepick;
okay4: AR=dm(inp2surrzerosptr);
    I6=AR;
    M6=1;
    AR=dm(I6, M6);
    dm(inp2end)=AR;
    AR=dm(I6, M6);
    AR=AR+0;
    IF GT jump okay5;
        call badtap;
        jump donepick;
okay5: dm(inp2start)=AR;
    AR=AR+0;
    IF GT jump okay6;
        call badtap;
        jump donepick;
okay6: jump donepick;

p22fine:
    AR=dm(inp1surrzerosptr);
    I3=AR;
    M3=1;
    AR=dm(I3, M3);
    dm(inp1end)=AR;
    AR=AR+0;
    IF GT jump okay7;
        call badtap;
        jump donepick;
okay7: AR=dm(I3, M3);
    dm(inp1start)=AR;
    AR=AR+0;
    IF GT jump okay8;
        call badtap;
        jump donepick;
okay8: AR=dm(inp2surrzerosptr);
    AR=AR+1;
    I6=AR;
    M6=1;
    AR=dm(I6, M6);
    dm(inp2end)=AR;
    AR=AR+0;
    IF GT jump okay9;
        call badtap;
        jump donepick;
okay9: AR=dm(I6, M6);
    dm(inp2start)=AR;
    AR=AR+0;
    IF GT jump okay10;
        call badtap;
okay10: jump donepick;

p21fine:
    AR=dm(I4, M4);
    AR=AR+0;
    IF GT jump p2sfine;

    AR=dm(inp1surrzerosptr);
    AR=AR+1;
    I3=AR;
    M3=1;
    AR=dm(I3, M3);
    dm(inp1end)=AR;
    AR=AR+0;
    IF GT jump okay11;
        call badtap;
        jump donepick;
okay11: AR=dm(I3, M3);
    dm(inp1start)=AR;
    AR=AR+0;
    IF GT jump okay12;
        call badtap;
        jump donepick;
okay12: AR=dm(inp2surrzerosptr);
    I6=AR;
    M6=1;
    AR=dm(I6, M6);
    dm(inp2end)=AR;
    AR=AR+0;
    IF GT jump okay13;
        call badtap;
        jump donepick;
okay13: AR=dm(I6, M6);
    dm(inp2start)=AR;
    AR=AR+0;
    IF GT jump okay14;
        call badtap;
okay14: jump donepick;

p2sfine:
    AR=dm(inp1peaksptr);
    AR=AR+2;
    I1=AR;
    AR=dm(inp2peaksptr);
    AR=AR+2;
    I4=AR;
    M1=0;
    M4=0;

    AR=dm(inp1surrzerosptr);
    I3=AR;
    M3=1;
    AR=dm(I3, M3);
    dm(inp1end)=AR;
    AR=AR+0;
    IF GT jump okay15;
        call badtap;
        jump donepick;
okay15: AR=dm(I3, M3);
    AR=dm(I3, M3);
    dm(inp1start)=AR;
    AR=AR+0;
    IF GT jump okay16;
        call badtap;
        jump donepick;
okay16: AR=dm(inp2surrzerosptr);
    I6=AR;
    M6=1;
    AR=dm(I6, M6);
    dm(inp2end)=AR;
    AR=AR+0;
    IF GT jump okay17;
        call badtap;
        jump donepick;
```

```
okay17: AR=dm(I6, M6);
    AR=dm(I6, M6);
    dm(inp2start)=AR;
    AR=AR+0;
    IF GT jump okay18;
        call badtap;
        jump donepick;
okay18: AR=dm(I1, M1);
    AR=AR+0;
    IF GT jump p31fine;

    AR=dm(I4, M4);
    AR=AR+0;
    IF LE jump donepick;

    AR=dm(inp2surrzerosptr);
    AR=AR+1;
    I6=AR;
    M6=1;
    AR=dm(I6, M6);
    dm(inp2end2)=AR;
    AR=AR+0;
    IF GT jump okay19;
        call badtap;
        jump donepick;
okay19: AR=dm(I6, M6);
    AR=dm(I6, M6);
    dm(inp2start2)=AR;
    AR=AR+0;
    IF GT jump okay20;
        call badtap;
        jump donepick;
okay20: AR=1;
    dm(inp2num)=AR;
    jump donepick;

p31fine:   AR=dm(I4, M4);
    AR=AR+0;
    IF GT jump p3sfine;
    AR=dm(inp1surrzerosptr);
    AR=AR+1;
    I3=AR;
    M3=1;
    AR=dm(I3, M3);
    dm(inp1end2)=AR;
    AR=AR+0;
    IF GT jump okay21;
        call badtap;
        jump donepick;
okay21: AR=dm(I3, M3);
    AR=dm(I3, M3);
    dm(inp1start2)=AR;
    AR=AR+0;
    IF GT jump okay22;
        call badtap;
        jump donepick;
okay22: AR=1;
    dm(inp1num)=AR;
    jump donepick;

p3sfine:   AR=dm(inp1surrzerosptr);
    AR=AR+1;
    I3=AR;
    M3=1;
    AR=dm(I3, M3);
    dm(inp1end2)=AR;
    AR=AR+0;
    IF GT jump okay23;
        call badtap;
        jump donepick;
okay23: AR=dm(I3, M3);
    AR=dm(I3, M3);
    dm(inp1start2)=AR;
    AR=AR+0;
    IF GT jump okay24;
        call badtap;
        jump donepick;
okay24: AR=1;
    dm(inp1num)=AR;

    AR=dm(inp2surrzerosptr);
    AR=AR+1;
    I6=AR;
    M6=1;
    AR=dm(I6, M6);
    dm(inp2end2)=AR;
    AR=AR+0;
    IF GT jump okay25;
        call badtap;
        jump donepick;
okay25: AR=dm(I6, M6);
    AR=dm(I6, M6);
    dm(inp2start2)=AR;
    AR=AR+0;
    IF GT jump okay26;
        call badtap;
        jump donepick;
okay26: AR=1;
    dm(inp2num)=AR;

donepick:  nop;

RTS;


copycorrinputs:

    ay0=dm(inp1ptr);
    AR=dm(inp1startfinal);
    AR=AR+ay0;
    AR=AR-1;
    I5=AR;
    M5=1;
    L5=0;

    ay0=dm(inp2ptr);
    AR=dm(inp2startfinal);
    AR=AR+ay0;
    AR=AR-1;
    I6=AR;
    M6=1;
    L6=0;

    I7=^chisqinput1;
    M7=1;
    L7=0;
    I0=^chisqinput2;
    M0=1;
    L0=0;

    ay0=dm(inp1startfinal);
    ax0=dm(inp1endfinal);

    AR=ax0-ay0;
    AR=AR+1;
    ay1=AR;
    ay0=dm(inp2startfinal);
    ax0=dm(inp2endfinal);
    AR=ax0-ay0;
    AR=AR+1;
    ax1=AR;
    AR=ax1-ay1;

    IF GE jump secondlonger2;
    dm(corrlength)=ay1;
    AR=ax1;
    AR=AR+0;
    IF GT jump cntrokay2;
        call badtap;
        jump donecopyinput;
cntrokay2: CNTR=ax1;

    DO copyfirst1 UNTIL CE;
        AR=dm(I5, M5);
        dm(I7, M7)=AR;
        AR=dm(I6, M6);
copyfirst1: dm(I0, M0)=AR;

    AR=ax1-ay1;
    AR=-AR;
    IF GT jump cntrokayn;
        call badtap;
        jump donecopyinput;
cntrokayn: CNTR=AR;
    DO copyrest1 UNTIL CE;
        AR=dm(I5, M5);
        dm(I7, M7)=AR;
        AR=0;
copyrest1: dm(I0, M0)=AR;

    jump donecopyinput;
secondlonger2: dm(corrlength)=ax1;

    AR=ay1;
    AR=AR+0;
    IF GT jump cntrokay3;
        call badtap;
        jump donecopyinput;
cntrokay3: CNTR=ay1;
    DO copyfirst2 UNTIL CE;
        AR=dm(I5, M5);
        dm(I7, M7)=AR;
        AR=dm(I6, M6);
copyfirst2: dm(I0, M0)=AR;
    AR=ax1-ay1;
    IF LE jump donecopyinput;
    CNTR=AR;
    DO copyrest2 UNTIL CE;
        AR=0;
        dm(I7, M7)=AR;
        AR=dm(I6, M6);
copyrest2: dm(I0, M0)=AR;

donecopyinput: nop;

RTS;

correlate:
    IMASK=0x0006;
    call copycorrinputs;
    I7=^chisqinput1;
    L7=0;
    I5=^chisqinput2;
    L5=0;
    M5=1;

    I6=^xcorrhi;
    L6=0;
    M6=1;

    I3=^xcorrlo;
    L3=0;
    M3=1;

    AR=dm(corrlength);
    AR=AR-1;
    ay0=AR;

    AR=I7;
    AR=AR+ay0;
    I7=AR;
    M7=-1;

    I2=1;
    M2=1;
    L2=0;

    M0=1;
    M4=1;

    AR=DM(corrlength);
    AR=AR+0;
    IF GT jump cntrokay5;
        call badtap;
        jump donecorrelate;
cntrokay5: CNTR=AR;


corrhalf1:
    DO corr_loop1 UNTIL CE;
    I0=I7;
    I4=I5;
    AR=I2;
    AR=AR+0;
    IF GT jump cntrfine19;
    AR=1;
cntrfine19: CNTR=I2;
    MR=0;
    DO data_loop1 UNTIL CE;
        MY0=DM(I4, M4);
        MX0=DM(I0, M0);
data_loop1: MR=MR+MX0*MY0(SS);
    MX0=DM(I7, M7);
    MX0=DM(I2, M2);
    SR=LSHIFT MR2 BY 11 (HI);
    SR=SR OR LSHIFT MR1 BY -5 (HI);
    SR=SR OR LSHIFT MR0 BY -5 (LO);
    DM(I3, M3)=SR0;
corr_loop1: DM(I6, M6)=SR1;
    nop;

    M2=-1;
    AR=DM(corrlength);
    AR=AR-1;
    IF GT jump cntrokay6;
        call badtap;
        jump donecorrelate;
cntrokay6: CNTR=AR;
    AR=I2;
    AR=AR-2;
    I2=AR;
    M7=1;
    MX0=DM(I7, M7);
    MX0=DM(I5, M5);

corrhalf2:
    DO corr_loop2 UNTIL CE;
    I0=I7;
    I4=I5;
    AR=I2;
    AR=AR+0;
    IF GT jump cntrfine22;
    AR=1;
cntrfine22: CNTR=I2;
    MR=0;
    DO data_loop2 UNTIL CE;
        MY0=DM(I4, M4);
        MX0=DM(I0, M0);
data_loop2: MR=MR+MX0*MY0(SS);
    MX0=DM(I5, M5);
    MX0=DM(I2, M2);
    SR=LSHIFT MR2 BY 11 (HI);
    SR=SR OR LSHIFT MR1 BY -5 (HI);
    SR=SR OR LSHIFT MR0 BY -5 (LO);
    DM(I3, M3)=SR0;
corr_loop2: DM(I6, M6)=SR1;

    AR=dm(corrlength);
    ay0=AR;
    AR=AR+ay0;
    AR=AR-1;
    dm(corrlength)=AR;
donecorrelate: nop;
RTS;


.ENDMOD;
```

C.2 C^{++} Code

```
// dataparser.cpp
//
// this will be used to parse the data taken from the tap tracker in the file parsetap.txt
// the format of the input is in the general form of output given by the DSP
// this program will basically open a file and do processing on it as did tapper.h
//
// Che King Leo
//
// version 0.4
// Added support to use the sensors in a "diamond" configuration
// Added Nearest Neighbor Algorithm for determining Tap Type
//
// version 0.3
// Includes support to change the coordinates of the sensor positions automatically.
// This is via the new hypfast routine, which runs in approximately lg() time.
//
// version 0.2
// Includes serial support. (see line 37, bool forSerial)
// This allows this file to be used to be integrated with real time apps using
// the Responsive Window with the DSP.
//
// version 0.1
// Only does data parsing of files taken from terminal windows from DSP
//


#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sys/timeb.h>
#include <winsock.h>
#include <fstream.h>
#include <math.h>

/**************** VARIABLES for SERIAL PORT **********/
static HANDLE hCom;
char serialPort[30] = "COM1";
int i, j, k;                  //for loops
unsigned char data[300];      //for storing data from the serial port
int pos=0;                    //pointer for data[]

/**************** SUBROUTINES ************************/
void comSetup();              //setup serial port
                              // hyp() has become obsolete...
void hypfast();               //finds intersection of points given corr13, corr24,
                              //and speed u via guessing heuristic
void hypbrute();              //brute force version of hypfast
void hyppreferred();          //method used to find intersection of hyperbolas
long getnumber(unsigned char*); //parses serial data from DSP
unsigned char* serialRead(); //reads data from serial port
void getTapType(double freq1, double freq2, double freq3, double freq4);
double getDistance(double x1, double y1, double x2, double y2);
double getMinimum(double x1, double x2, double x3, double x4);
double getMaximum(double x1, double x2, double x3, double x4);
double getMaximum(double x1, double x2, double x3, double x4, double x5, double x6);
void hypbrute(double x1, double y1, double x2, double y2, double x3,
            double y3, double x4, double y4, double corr13, double corr24);
void getClosestPoint(double x1, double y1, double x2, double y2,
                   double x3, double y3, double x4, double y4, double x, double y);
unsigned char* read();
void run();
void initializeGlobals();
void transposeSensors();     // transposes sensors for diamond configuration for use with hypfast
void untransposeOutput();    // untransposes the output so it is correct


/***************** VARIABLES ***********************/
bool forSerial = 0;   // toggles serial reading with data file parsing.
                      // 1 = serial. 0 = data file parsing.
bool forDiamond = 0;  // toggles configuration of four sensors for transposeSensors use

int globalpos;
int datalength;
char mypointer[1000000];
double xposition;      // resulting xposition from hyp()
double yposition;      // resulting yposition from hyp()
double NUMBEROFKNUCKLECALS=81;
double NUMBEROFMETALCALS=62;

double savex1, savex2, savex3; // vars for storing xpositions from hyp()
double savey1, savey2, savey3; // vars for storing ypositions from hyp()

double incorr13, incorr24; // actual variables used to pass
```

```
                        // corr13 and corr24 to hyp()
double corr13, corr24;   // variable inputs for hyp()
double corr132, corr242; // secondary positions for hyp()

double zeroxings;      // sum total of zero crossings over 480 samples
double humpwidth;      // resulting avg width of waveforms (NOT period)
double metalu, knuckleu; // variables to change
double u;              // speed for hyp()
                       // for now, metal tap is 1300, knuckle is 550.
double maxfreqdiff;    // maximum frequency difference between sensors before
                       // knock is considered to be tainted with cell phone noise

int ss1, ss2, ss3, ss4; // integrated amplitude for sensors.
int ms1, ms2, ms3, ms4; // maximum magnitude for sensors.
int ms5, ms6;          // bang amplitude and clap detector amplitude

int proxsensor;        // if cannot be determined, proxsensor will say
                       // which sensor is closest by guess.

int badtap;            // if == 1, means tap was bad (from DSP)
                       // (formerly called tapflag)
int xboundaryflag;     //  0 means x coordinate is valid;
                       // -1 means to the right of all sensors;
                       //  1 means to the left
int yboundaryflag;     // like above, but for y coordinates
                       // -1 means below all sensors; 1 means above

double precisionradius; // radius of certainty...
                       // don't know if this is a good variable to use
double accuracy;       // number that tells how good it is.
                       // lower is better.

int cpl13, cpl132;     // correlation peak location btwn 1 and 3
                       // (highest, 2nd highest, resp.)
int cpl24, cpl242;     // correlation peak location btwn 2 and 4
                       // (highest, 2nd highest, resp.)
double cpa13, cpa132;  // correlation peak amplitude btwn 1 and 3
                       // (highest, 2nd highest, resp.)
double cpa24, cpa242;  // correlation peak amplitude btwn 2 and 4
                       // (highest, 2nd highest, resp.)
double cha13, cha132;  // chi-square peak amplitudes
double cha24, cha242;
int typetriggered;     // type of tap
                       // double, knuckle, metal, bang, clap = 0,1,2,3,4
double sensor1x, sensor1y; // sensor positions
double sensor2x, sensor2y;
double sensor3x, sensor3y;
double sensor4x, sensor4y;

double sensor1xT, sensor1yT; // sensor positions, transposed
double sensor2xT, sensor2yT;
double sensor3xT, sensor3yT;
double sensor4xT, sensor4yT;
double ptx, pty, counter;// global vars for getClosestPoint
double avgX, avgY;     // mean point for center of glass
double mycounter;

double kdata[81][6] =    {{329,375,12,7,6,6},
                          {330,412,12,8,15,10},
                          {356,435,9,8,7,7},
                          {391,465,10,9,6,5},
                          {420,462,8,7,6,6},
                          {394,447,10,9,7,6},
                          {374,414,12,13,11,7},
                          {349,399,15,8,17,11},
                          {356,367,16,8,8,14},
                          {372,387,13,10,8,9},
                          {396,408,11,8,6,7},
                          {419,432,9,9,6,9},
                          {440,444,9,8,7,6},
                          {462,432,9,9,7,6},
                          {446,424,8,8,7,6},
                          {415,388,9,8,7,6},
                          {393,365,13,13,9,11},
                          {373,339,9,10,12,9},
                          {391,317,10,10,6,9},
                          {402,337,12,10,8,10},
                          {424,368,10,9,12,9},
                          {460,379,11,8,7,6},
                          {481,402,9,7,7,8},
                          {500,412,8,6,6,6},
                          {475,372,7,6,6,8},
                          {419,338,11,7,6,11},
                          {401,401,10,9,7,7},
                          {338,389,14,10,22,16},
                          {338,407,14,11,22,11},
                          {362,428,22,13,24,21},
                          {381,454,12,13,13,8},
                          {398,472,13,12,17,11},
                          {416,460,13,11,7,12},
                          {406,443,12,12,16,10},
                          {382,431,17,14,11,11},
                          {354,403,13,15,9,20},
                          {348,368,11,12,6,9},
                          {374,393,15,11,15,15},
                          {400,415,15,21,18,13},
                          {423,444,10,14,10,20},
                          {438,437,14,13,25,25},
                          {432,417,12,17,25,20},
                          {406,405,14,13,10,9},
                          {305,255,13,11,4,4},
                          {363,355,14,11,8,11},
                          {377,339,11,11,13,9},
                          {391,347,12,10,6,7},
                          {418,374,12,10,11,7},
                          {428,393,14,13,21,18},
                          {474,401,9,8,14,11},
                          {460,414,14,15,20,21},
                          {471,394,18,13,17,17},
                          {448,367,9,12,17,25},
                          {422,331,12,10,10,11},
                          {346,388,3,5,2,4},
                          {317,410,11,10,13,8},
                          {366,424,12,11,9,10},
                          {381,446,10,10,14,6},
                          {410,470,8,4,7,4},
                          {387,437,11,9,11,8},
                          {370,412,11,8,7,7},
                          {354,390,7,6,5,8},
                          {356,376,7,6,5,12},
                          {370,390,8,8,6,8},
                          {397,416,7,8,9,7},
                          {414,435,5,5,5,8},
                          {425,436,10,7,7,8},
                          {446,428,4,4,6,4},
                          {434,411,5,6,5,8},
                          {403,404,6,6,4,5},
                          {391,375,7,6,5,4},
                          {381,358,7,5,9,4},
                          {390,357,7,6,4,5},
                          {419,383,9,7,6,6},
                          {450,401,8,7,6,10},
                          {459,411,9,8,6,6},
                          {479,386,6,5,5,4},
                          {443,381,8,7,6,4},
                          {430,352,12,6,9,6},
                          {392,308,4,4,4,3},
                          {391,402,8,7,4,7}};

 double mdata[62][6] =    {{366,392,33,30,28,27},
                          {372,408,33,29,29,30},
                          {380,416,34,33,31,36},
                          {390,426,32,29,30,30},
                          {396,434,26,29,30,27},
                          {411,431,22,26,30,31},
                          {403,423,31,26,30,32},
                          {393,411,34,34,32,33},
                          {381,401,33,32,30,30},
                          {382,388,31,32,30,25},
                          {375,383,33,32,29,28},
                          {389,394,33,33,30,31},
                          {400,408,33,30,34,31},
                          {409,418,28,31,31,34},
                          {415,422,25,30,33,33},
                          {425,418,31,29,33,33},
                          {413,407,33,34,32,31},
                          {401,398,33,32,32,33},
                          {392,383,33,35,31,29},
                          {388,376,32,33,30,30},
                          {398,381,34,33,31,30},
                          {409,395,33,31,34,32},
                          {424,408,31,33,33,31},
                          {435,407,25,27,30,32},
                          {428,400,29,32,32,31},
                          {409,379,30,33,33,30},
                          {397,370,26,32,30,28},
                          {405,360,29,32,30,27},
                          {416,372,30,34,32,29},
                          {427,389,28,31,33,29},
                          {437,396,23,28,31,26},
                          {360,383,29,24,27,27},
                          {364,404,32,27,25,28},
                          {369,411,34,26,27,32},
                          {387,419,29,29,29,32},
                          {389,432,33,30,28,32},
                          {404,435,28,25,30,33},
                          {398,425,31,29,29,30},
                          {387,410,36,31,30,32},
                          {376,399,32,31,29,29},
                          {376,385,33,32,29,28},
                          {381,395,30,30,30,29},
                          {391,409,32,32,31,32},
                          {403,418,28,31,31,36},
                          {411,424,29,26,27,34},
                          {423,418,28,31,32,33},
                          {418,412,27,30,31,33},
                          {404,397,31,34,33,31},
                          {392,388,34,32,31,31},
                          {378,379,25,32,28,29},
                          {386,366,24,32,29,27},
                          {402,385,29,33,31,30},
                          {409,393,29,32,32,32},
                          {428,404,30,26,32,30},
                          {436,409,23,27,32,30},
                          {436,397,28,31,32,28},
                          {417,387,28,33,33,30},
                          {404,372,29,33,29,28},
                          {403,355,30,31,28,27},
                          {415,366,26,30,31,31},
                          {427,389,28,31,32,27},
                          {437,398,26,28,32,27}};

double distance, distance1, distance2, distance3, distance4, distance5;
double type, type1, type2, type3, type4, type5;
double scale01off, scale01, scale2345;

bool coeffSet13;       // decides to initialize hyp vars as either
                       //for sensors 1 and 3 or 2 and 4
                       // 1 and 3 == TRUE; 2 and 4 == FALSE;


//double timedifference;
//clock_t start, finish;

void main() {
    cout << "initializing globals...\n";
    initializeGlobals();

    if (forSerial == TRUE) {
        cout << "Using serial port.\n";
        comSetup();
        cout << "\n";
        while(1) {
            serialRead();
            //cout << "gotdata \n"<< endl;
            run();
        }
    } else {
        cout << "Using datafile parsetap.dat.\n";
        ifstream newdata("parsetap.dat",ios::in);
        if (!newdata) {
                cerr << "File could not be opened" <<endl;
                exit(1);
        }
        newdata >> mypointer;
        datalength = strlen(mypointer);
        //cout << "gotpointer\n";
        while (globalpos < datalength) {
            read();
            run();
        }
        //cout << strlen(mypointer) << endl;
    }
}

void initializeGlobals(){
    // initialize globals

    /* identity data */

    sensor1x = 158.0;
    sensor1y = 117.0;
    sensor2x = 158.0;
    sensor2y = 0.0;
    sensor3x = 0.0;
    sensor3y = 0.0;
    sensor4x = 0.0;
    sensor4y = 117.0;


    /* 1CC data */
    // diagonal formation
    /*
    sensor1x = 67.0;
    sensor1y = 129.0;
    sensor2x = 10.0;
    sensor2y = 77.0;
    sensor3x = 72.0;
    sensor3y = 12.5;
    sensor4x = 129.0;
    sensor4y = 76.0;
    */
    // rectangular formation
    /*
    sensor1x = 25.0;
    sensor1y = 116.0;
    sensor2x = 25.0;
    sensor2y = 25.0;
    sensor3x = 116.0;
    sensor3y = 25.0;
    sensor4x = 116.0;
    sensor4y = 116.0;
    */

    knuckleu = 696;
    metalu = 1500;
    maxfreqdiff = 4;
    avgX = (sensor1x + sensor2x + sensor3x + sensor4x) / 4.0;
    avgY = (sensor1y + sensor2y + sensor3y + sensor4y) / 4.0;
    scale01off = 250.0;
    scale01 = 300.0;
    scale2345 = 16.0;


    for (int i=0; i<NUMBEROFKNUCKLECALS; i++) {
        kdata[i][0] = (kdata[i][0]-scale01off)/scale01;
        kdata[i][1] = (kdata[i][1]-scale01off)/scale01;
        kdata[i][2] = kdata[i][2]/scale2345;
        kdata[i][3] = kdata[i][3]/scale2345;
        kdata[i][4] = kdata[i][4]/scale2345;
        kdata[i][5] = kdata[i][5]/scale2345;
    }
    for (i=0; i<NUMBEROFMETALCALS; i++) {
        mdata[i][0] = (mdata[i][0]-scale01off)/scale01;
        mdata[i][1] = (mdata[i][1]-scale01off)/scale01;
        mdata[i][2] = mdata[i][2]/scale2345;
        mdata[i][3] = mdata[i][3]/scale2345;
        mdata[i][4] = mdata[i][4]/scale2345;
        mdata[i][5] = mdata[i][5]/scale2345;
    }


    /* AG window data
    sensor1x = 169.0;
    sensor1y = 130.0;
    sensor2x = 168.0;
    sensor2y = 0.0;
    sensor3x = 0.0;
    sensor3y = 0.0;
    sensor4x = 0.0;
    sensor4y = 133.0;

    knuckleu = 1200;
    metalu = 2000;
    maxfreqdiff = 4; */
/*
    incorr13 = 392;
    incorr24 = 384;
    u = knuckleu;
    hyppreferred();
    cout << xposition << ", " << yposition << endl;
    cout << avgX << ", " << avgY << endl;
*/
}


void getClosestPoint(double x1, double y1, double x2, double y2,
                     double x3, double y3, double x4, double y4, double x, double y) {
    double distance, temp;
    counter = 1;
    distance = (x1 - x) * (x1 - x) + (y1 - y) * (y1 - y);
    ptx = x1;
    pty = y1;
    temp = (x2 - x) * (x2 - x) + (y2 - y) * (y2 - y);
    if (temp < distance) {
        ++counter;
        distance = temp;
        ptx = x2;
        pty = y2;
    }
    temp = (x3 - x) * (x3 - x) + (y3 - y) * (y3 - y);
    if (temp < distance) {
        ++counter;
        distance = temp;
        ptx = x3;
        pty = y3;
    }
    temp = (x4 - x) * (x4 - x) + (y4 - y) * (y4 - y);
    if (temp < distance) {
        ++counter;
        distance = temp;
        ptx = x4;
        pty = y4;
    }
}


unsigned char* read() {

    unsigned char buffer;
    data[0] = 0;
    pos = 0;
    while (1) {
        buffer = mypointer[globalpos++];
        if (pos > datalength) {
            return data;
        }
        if (pos < 300) {

            data[pos++] = buffer;

            if (pos > 5) {
                if (!isalnum((int)data[pos-1]) && !isalnum((int)data[pos-2]) &&
                                                    !isalnum((int)data[pos-3])){
                    //cout << "data go parse" << endl;
                    return data; // data in data, pos in pos (globals)
                }
            }
        }
    }
}


void run() {

    long temp; // temp variable for getting data from serial port.
    long temp1, temp2, temp3, temp4; // for getting rid of double taps, accuracy
    double freq1, freq2, freq3, freq4; // records the number of zeroxings for sensors 1-4
    double freqdiff12, freqdiff13, freqdiff14, freqdiff23, freqdiff24, freqdiff34;
    double doubletemp1, doubletemp2;
    typetriggered = 1; // reset this number!

    // sum all the magnitudes
    // veto the DSP if it's stupid on type of tap
    temp = getnumber(&data[20]);
    freq1 = temp;
    if (temp >= 10 && data[150] != 'M') {
        //data[150] = 'N';
    }
    zeroxings = temp;
    temp = getnumber(&data[45]);
    freq2 = temp;
    if (temp >= 10 && data[150] != 'M') {
        //data[150] = 'N';
    }
    zeroxings += temp;
    temp = getnumber(&data[70]);
    freq3 = temp;
    if (temp >= 10 && data[150] != 'M') {
        //data[150] = 'N';
    }
    zeroxings += temp;
    temp = getnumber(&data[95]);
    freq4 = temp;
    if (temp >= 10 && data[150] != 'M') {
        //data[150] = 'N';
    }
    //data[150] = 'N';
    zeroxings += temp;
    humpwidth = 480.0/zeroxings;
    if (zeroxings > 37 && data[150] != 'M') {
        //data[150] = 'N';
    }

    //sum and max magnitude of waveforms
```

```
ss1 = getnumber(&data[6]);
ms1 = getnumber(&data[13]);
ss2 = getnumber(&data[31]);
ms2 = getnumber(&data[38]);
ss3 = getnumber(&data[56]);
ms3 = getnumber(&data[63]);
ss4 = getnumber(&data[81]);
ms4 = getnumber(&data[88]);
ms5 = getnumber(&data[113]);
ms6 = getnumber(&data[138]);

if (data[150] == 'M' || data[150] == 'N') {
                         // means DSP thinks a metal tap has occurred
    typetriggered = 2;
    u = metalu;

    if (data[150] == 'M') {

        temp = getnumber(&data[152]);
        temp1 = temp;
        corr13 = 400 + temp;
        temp = getnumber(&data[157]);
        corr24 = 400 + temp;
        temp2 = temp;
        temp = getnumber(&data[162]);
        corr13 -= temp;
        temp3 = temp;
        temp = getnumber(&data[167]);
        corr24 -= temp;
        temp4 = temp;
    }  else {
        //cout << data[150];
        //cout << " knuckle converted to metal!\n";
        temp = getnumber(&data[254]);
        temp1 = temp;
        corr13 = 400 + temp;
        temp = getnumber(&data[259]);
        corr24 = 400 + temp;
        temp2 = temp;
        temp = getnumber(&data[264]);
        corr13 -= temp;
        temp3 = temp;
        temp = getnumber(&data[269]);
        corr24 -= temp;
        temp4 = temp;
        data[150] = 'M';

        temp = getnumber(&data[169]);
        corr13 = 400 + temp;
        temp = getnumber(&data[174]);
        corr132 = 400 + temp;
        temp = getnumber(&data[219]);
        corr24 = 400 + temp;
        temp = getnumber(&data[224]);
        corr242 = 400 + temp;

    }
    if ((temp1 < 60 || temp2 < 60 || temp3 < 60 || temp4 < 60) && ms5 < 256 ) {
        //printf("dying 1\n");
        typetriggered = 0;
        xposition = 0;
        yposition = 0;
        goto ENDOFPROCESS;
    }

    badtap = 0;
} else { // means DSP thinks a knuckle tap has occurred
    typetriggered = 1;
    temp = getnumber(&data[254]);
    temp1 = temp;
    temp = getnumber(&data[259]);
    temp2 = temp;
    temp = getnumber(&data[264]);
    temp3 = temp;
    temp = getnumber(&data[269]);
    temp4 = temp;
    if ((temp1 < 60 || temp2 < 60 || temp3 < 60 || temp4 < 60) && ms5 < 256 ) {
        typetriggered = 0;
        xposition = 0;
        yposition = 0;
        goto ENDOFPROCESS;
    }

    u = knuckleu;

    badtap = getnumber(&data[274]);
```

```
temp = getnumber(&data[169]);
corr13 = 400 + temp;
temp = getnumber(&data[174]);
corr132 = 400 + temp;
temp = getnumber(&data[219]);
corr24 = 400 + temp;
temp = getnumber(&data[224]);
corr242 = 400 + temp;

// correlation peak locations and amplitudes
cpl13 = getnumber(&data[169]);
cpl132 = getnumber(&data[174]);
cpl24 = getnumber(&data[219]);
cpl242 = getnumber(&data[224]);
// neither word can ever be negative
temp = getnumber(&data[186]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa13 = temp;
temp = getnumber(&data[181]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa13 += temp * 65536 ;

temp = getnumber(&data[197]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa132 = temp;
temp = getnumber(&data[192]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa132 += temp * 65536;

temp = getnumber(&data[236]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa24 = temp;
temp = getnumber(&data[231]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa24 += temp * 65536;

temp = getnumber(&data[247]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa242 = temp;
temp = getnumber(&data[242]);
if (temp < 0) {
   temp = temp + 65535;
}
if (temp == 32768) {
   temp = 0;
}
cpa242 += temp * 65536;

// get chi-squared amplitudes
cha13 = getnumber(&data[157]);
cha132 = getnumber(&data[162]);
cha24 = getnumber(&data[207]);
```

```
        cha242 = getnumber(&data[212]);

    }


    // determine proxsensor value
    // use the metal taps, which are so conveniently stored in temp1-4

    proxsensor = 1;
    temp = temp1;
    if (temp2 > temp) {
        proxsensor = 2;
        temp = temp2;
    }
    if (temp3 > temp) {
        proxsensor = 3;
        temp = temp3;
    }
    if (temp4 > temp) {
        proxsensor = 4;
    }

    if (typetriggered == 1 || typetriggered == 2 || typetriggered == 3) {

        // do a nearest neighbor search
        incorr13 = 400 + temp1 - temp3;
        incorr24 = 400 + temp2 - temp4;

        hyppreferred();       //hyperbola estimate based on metal tap
        savex2 = xposition;
        savey2 = yposition;

        incorr13 = corr13;    //hyperbola estimate based on chi-square cross-corr
        incorr24 = corr24;
        savex1 = 0;
        savey1 = 0;
        hyppreferred();
        savex1 = xposition;
        savey1 = yposition;



        if (getDistance(savex2, savey2, xposition, yposition) < 10) {
            xposition = (xposition + savex2) / 2.0;
            yposition = (yposition + savey2) / 2.0;
        } else {
            xposition = savex2;
            yposition = savey2;
        }

    }

ENDOFPROCESS:
    mycounter++;
    // only print out taps that were good
    if (typetriggered != 0) {
        cout << typetriggered << "\t";
        cout << savex2 << "\t" << savey2 << "\t" << xposition << "\t" << yposition << "\t" << savex1 << "\t" << savey1 << "\t";
        cout << freq1 + freq2 + freq3 + freq4 << "\t" << mycounter << endl;
    }


}

void getTapType(double freq1, double freq2, double freq3, double freq4) {

        double distance1, distance2, distance3, distance4, distance5;
        int type1, type2, type3, type4, type5;
        distance1 = 10000000;
        distance2 = 10000001;
        distance3 = 10000002;
        distance4 = 10000003;
        distance5 = 10000004;
        type1 = 0;
        type2 = 0;
        type3 = 0;
        type4 = 0;
        type5 = 0;


        for (i=0; i<NUMBEROFMETALCALS; i++) {
            // calculate nearest neighbors for metal calibration points
            distance = pow(((incorr13-scale01off)/scale01-mdata[i][0]), 2) +
                        pow(((incorr24-scale01off)/scale01-mdata[i][1]), 2) +
                        pow((freq1/scale2345-mdata[i][2]), 2) + pow((freq2/scale2345-mdata[i][3]), 2) +
                        pow((freq3/scale2345-mdata[i][4]), 2) + pow((freq4/scale2345-mdata[i][5]), 2);
            distance = sqrt(distance);
        if (distance < distance1) {
            distance5 = distance4;
            distance4 = distance3;
            distance3 = distance2;
            distance2 = distance1;
            type5 = type4;
            type4 = type3;
            type3 = type2;
            type2 = type1;
            distance1 = distance;
            type1 = -1;
        } else if (distance < distance2) {
            distance5 = distance4;
            distance4 = distance3;
            distance3 = distance2;
            type5 = type4;
            type4 = type3;
            type3 = type2;
            distance2 = distance;
            type2 = -1;
        } else if (distance < distance3) {
            distance5 = distance4;
            distance4 = distance3;
            type5 = type4;
            type4 = type3;
            distance3 = distance;
            type3 = -1;
        } else if (distance < distance4) {
            distance5 = distance4;
            type5 = type4;
            distance4 = distance;
            type4 = -1;
        } else if (distance < distance5) {
            distance5 = distance;
            type5 = -1;
        }
    }

    for (i=0; i<NUMBEROFKNUCKLECALS; i++) {
        // calculate nearest neighbors for knuckle
        distance = pow(((incorr13-scale01off)/scale01-kdata[i][0]), 2) +
                    pow(((incorr24-scale01off)/scale01-kdata[i][1]), 2) +
                        pow((freq1/scale2345-kdata[i][2]), 2) + pow((freq2/scale2345-kdata[i][3]), 2) +
                        pow((freq3/scale2345-kdata[i][4]), 2) + pow((freq4/scale2345-kdata[i][5]), 2);
        distance = sqrt(distance);
        if (distance < distance1) {
            distance5 = distance4;
            distance4 = distance3;
            distance3 = distance2;
            distance2 = distance1;
            type5 = type4;
            type4 = type3;
            type3 = type2;
            type2 = type1;
            distance1 = distance;
            type1 = 1;
        } else if (distance < distance2) {
            distance5 = distance4;
            distance4 = distance3;
            distance3 = distance2;
            type5 = type4;
            type4 = type3;
            type3 = type2;
            distance2 = distance;
            type2 = 1;
        } else if (distance < distance3) {
            distance5 = distance4;
            distance4 = distance3;
            type5 = type4;
            type4 = type3;
            distance3 = distance;
            type3 = 1;
        } else if (distance < distance4) {
            distance5 = distance4;
            type5 = type4;
            distance4 = distance;
            type4 = 1;
        } else if (distance < distance5) {
            distance5 = distance;
            type5 = 1;
        }

    }

    type = type1 + type2 + type3 + type4 + type5;
    if (type < 0.0) {
        u = metalu;
```

```
            typetriggered = 2;
        } else if (type > 0.0){
            u = knuckleu;
            typetriggered = 1;
        } else {
            if (type1 == -1) {
                u = metalu;
                typetriggered = 2;
            } else {
                u = knuckleu;
                typetriggered = 1;
            }
        }
}





long getnumber(unsigned char* mypointer) {
    int finalint = 0;
    const char * myowntemp = ((const char *) mypointer);
    long myx = strtol(myowntemp, NULL, 16);
    if (myx > 32767) {
        myx = myx - 65535;
    }
    return myx;
}

void hyppreferred() {
    // Preferred method for finding intersection of hyperbolas
        transposeSensors();
        hypfast();
        untransposeOutput();

}

void transposeSensors() {
    if (forDiamond == 1) {
        // center on zero and rotate coordinates by w radians
        //  x' = x cos(w) - y sin(w)
        //  y' = x sin(w) + y cos(w)
        //  in this incarnation we will rotate by 45 degrees
        sensor1xT = (sensor1x - sensor1y - avgX + avgY)/ sqrt(2.0) + avgX;
        sensor1yT = (sensor1x + sensor1y - avgX - avgY)/ sqrt(2.0) + avgY;
        sensor2xT = (sensor2x - sensor2y - avgX + avgY)/ sqrt(2.0) + avgX;
        sensor2yT = (sensor2x + sensor2y - avgX - avgY)/ sqrt(2.0) + avgY;
        sensor3xT = (sensor3x - sensor3y - avgX + avgY)/ sqrt(2.0) + avgX;
        sensor3yT = (sensor3x + sensor3y - avgX - avgY)/ sqrt(2.0) + avgY;
        sensor4xT = (sensor4x - sensor4y - avgX + avgY)/ sqrt(2.0) + avgX;
        sensor4yT = (sensor4x + sensor4y - avgX - avgY)/ sqrt(2.0) + avgY;
    } else {
        sensor1xT = sensor1x;
        sensor1yT = sensor1y;
        sensor2xT = sensor2x;
        sensor2yT = sensor2y;
        sensor3xT = sensor3x;
        sensor3yT = sensor3y;
        sensor4xT = sensor4x;
        sensor4yT = sensor4y;
    }
}

void untransposeOutput() {
    double xpositionTemp, ypositionTemp;
    xpositionTemp = xposition;
    ypositionTemp = yposition;
    if (forDiamond == 1) {
        xposition = (xpositionTemp + ypositionTemp - avgX - avgY)/ sqrt(2.0) + avgX;
        yposition = (-xpositionTemp + ypositionTemp + avgX - avgY)/ sqrt(2.0) + avgY;
    }
}

void hypfast() {
    // we want to minimize:
    // abs(sqrt((x-sensor3x)^2 + (y-sensor3y)^2) + sqrt((x-sensor4x)^2 +
    //(y-sensor4y)^2) - sqrt((x-sensor1x)^2 + (y-sensor1y)^2) -
    // sqrt((x-sensor2x)^2 + (y-sensor2y)^2) - (d1 + d2))
    // abs(sqrt((x-x3)^2 + (y-y3)^2) + sqrt((x-x4)^2 + (y-y4)^2) -
    //sqrt((x-x1)^2 + (y-y1)^2) - sqrt((x-x2)^2 + (y-y2)^2) - (d1 + d2))
    double w, d1, d2, x, y, xSET, ySET, xmin, ymin, xmax, ymax, toMinimize;
    double absmin, absmax, absmid1, absmid2;
    // assume positions x1, x4, x3, x2, clockwise starting from top left from inside view
    // assume xk + yk is smallest for k = 3
    // change the stupid names of the global sensor positions
    double x1, x2, x3, x4, y1, y2, y3, y4;
    double temp1, temp2;
```

```
x1 = sensor1xT;
y1 = sensor1yT;
x2 = sensor2xT;
y2 = sensor2yT;
x3 = sensor3xT;
y3 = sensor3yT;
x4 = sensor4xT;
y4 = sensor4yT;

//cout << x1 << x2 << x3 << x4 << y1 << y2 << y3 << y4<< "\n";
//cin >> w;

double m = 400;
w = 100.0 * u / 50000.0;
// this makes no sense, but it works... apparently i screwed up somewhere
d1 = w * (m - incorr13);
d2 = w * (m - incorr24);

// guess a point -- the center?
xSET = abs(x1 + x2 + x3 + x4)/ 4.0;
ySET = abs(y1 + y2 + y3 + y4)/ 4.0;
x = xSET;
y = ySET;
temp1 = abs(sqrt(pow((x-x3),2.0) + pow((y-y3),2.0)) - sqrt(pow((x-x1),2.0) + pow((y-y1),2.0)) - d1);
temp2 = abs(sqrt(pow((x-x4),2.0) + pow((y-y4),2.0)) - sqrt(pow((x-x2),2.0) + pow((y-y2),2.0)) - d2);
//cout << temp1 << " " << temp2 << "\n";
toMinimize = pow(temp1,2) + pow(temp2, 2);

for (int i=1; i<5; i++) {
//while (toMinimize > 1) {
    // minimize toMinimize w.r.t. y when x = XSET.
    // binary search for better minimum point ySET
    ymin = -1000;
    ymax = 1000;
    while (((ymax - ymin) > 0.5) && (ymax > 0)) {
        y = (ymin + ymax)/2.0;
        temp1 = fabs(sqrt(pow((x-x3),2.0) + pow((y-y3),2.0)) - sqrt(pow((x-x1),2.0) + pow((y-y1),2.0)) - d1);
        temp2 = fabs(sqrt(pow((x-x4),2.0) + pow((y-y4),2.0)) - sqrt(pow((x-x2),2.0) + pow((y-y2),2.0)) - d2);
        absmid1 = pow(temp1,2) + pow(temp2, 2);
        y = y + 0.1;
        temp1 = fabs(sqrt(pow((x-x3),2.0) + pow((y-y3),2.0)) - sqrt(pow((x-x1),2.0) + pow((y-y1),2.0)) - d1);
        temp2 = fabs(sqrt(pow((x-x4),2.0) + pow((y-y4),2.0)) - sqrt(pow((x-x2),2.0) + pow((y-y2),2.0)) - d2);
        absmid2 = pow(temp1,2) + pow(temp2, 2);
        //cout << "ymin:ymax are" << ymin << ":" << ymax<< "\t" << y << ": "<< absmid1 << " "<< absmid2 << "\n";
        if (fabs(absmid1 - absmid2) < 0.001) {
            break;
        } else if (absmid1 < absmid2) {
            ymax = y +0.01;
        } else {
            ymin = y;
        }
        /*if (absmid1 < absmid2) {
            ymax = y +0.01;
        } else if (absmid1 > absmid2) {
            ymin = y;
        } else {
            //break;
        }*/
    }
    //cout << "ymin:ymax are" << ymin << ":" << ymax<< "\n";
    ySET = (ymin + ymax) /2.0;
    y = ySET;

    // minimize toMinimize w.r.t. x when y = ySET.
    // binary search for better minimum point xSET
    xmin = -1000;
    xmax = 1000;
    while ((xmax - xmin) > 0.5 && (xmax > 0)) {

        x = (xmin + xmax)/2.0;
        temp1 = fabs(sqrt(pow((x-x3),2.0) + pow((y-y3),2.0)) - sqrt(pow((x-x1),2.0) + pow((y-y1),2.0)) - d1);
        temp2 = fabs(sqrt(pow((x-x4),2.0) + pow((y-y4),2.0)) - sqrt(pow((x-x2),2.0) + pow((y-y2),2.0)) - d2);
        absmid1 = pow(temp1,2) + pow(temp2, 2);
        x = x + 0.01;
        temp1 = fabs(sqrt(pow((x-x3),2.0) + pow((y-y3),2.0)) - sqrt(pow((x-x1),2.0) + pow((y-y1),2.0)) - d1);
        temp2 = fabs(sqrt(pow((x-x4),2.0) + pow((y-y4),2.0)) - sqrt(pow((x-x2),2.0) + pow((y-y2),2.0)) - d2);
        absmid2 = pow(temp1,2) + pow(temp2, 2);
        //cout << "xmin:xmax are" << xmin << ":" << xmax<< "\t" << x << ": "<< absmid1 << " "<< absmid2 << "\n";
        if (fabs(absmid1 - absmid2) < 0.001) {
            break;
        } else if (absmid1 < absmid2) {
            xmax = x +0.01;
        } else {
            xmin = x;
        }
        /*if (absmid1 < absmid2) {
                xmax = x +0.01;
            } else if (absmid1 > absmid2) {
                xmin = x;
            } else {
                break;
            }*/
        }
        xSET = (xmin + xmax) / 2.0;
        if ((absmid1 > toMinimize) || (absmid2 > toMinimize)) {
            break;
        }
        if ((xmax <= 0) || (ymax <= 0)) {
            break;
        }
        toMinimize = absmid1;
    }
    //cout << toMinimize << "\n";
    xposition = xSET;
    yposition = ySET;

}

void hypbrute() {
    // assumes point is inside grid
    double temp1, temp2, temp3 , temp4;
    double x1, x2, x3, x4, y1, y2, y3, y4;
    // uh.... a brute force search.
    double oldmin, min, x, y, xSET, ySET;
    double w, d1, d2, temp1a, temp2a;
    double m = 400;

    x1 = sensor1x;
    y1 = sensor1y;
    x2 = sensor2x;
    y2 = sensor2y;
    x3 = sensor3x;
    y3 = sensor3y;
    x4 = sensor4x;
    y4 = sensor4y;

    // get size of grid
    temp1 = getMinimum(x1, x2, x3, x4);
    temp2 = getMaximum(x1, x2, x3, x4);
    temp3 = getMinimum(y1, y2, y3, y4);
    temp4 = getMaximum(y1, y2, y3, y4);

    //corr13 = 400 - mytemp1 + mytemp3;
    //corr24 = 400 - mytemp2 + mytemp4;
    w = 100.0 * u / 50000.0;
    d1 = w * (incorr13 - m);
    d2 = w * (incorr24 - m);

    oldmin = 10000000;
    for (int i=(int) temp1; i < (int) temp2; i++) {
        for (int j=(int) temp3; j < (int) temp4; j++) {
            x = i;
            y = j;
            temp1a = fabs(-sqrt(pow((x-x4),2.0) + pow((y-y4),2.0)) + sqrt(pow((x-x2),2.0) + pow((y-y2),2.0)) - d2);
            temp2a = fabs(-sqrt(pow((x-x3),2.0) + pow((y-y3),2.0)) + sqrt(pow((x-x1),2.0) + pow((y-y1),2.0)) - d1);
            min = pow(temp1a, 2.0) + pow(temp2a, 2.0);
            if (min < oldmin) {
                oldmin = min;
                xSET = x;
                ySET = y;
            }
        }
    }
    //cout << xSET << "\t" << ySET << endl;

    xposition = xSET;
    yposition = ySET;


}

void hypbrute(double x1, double y1, double x2, double y2, double x3,
                double y3, double x4, double y4, double corr13, double corr24) {
    // assumes point is inside grid
    double temp1, temp2, temp3 , temp4;

    // uh.... a brute force search.
    double oldmin, min, x, y, xSET, ySET;
    double w, d1, d2, temp1a, temp2a;
    double m = 400;

    // get size of grid
    temp1 = getMinimum(x1, x2, x3, x4);
    temp2 = getMaximum(x1, x2, x3, x4);
    temp3 = getMinimum(y1, y2, y3, y4);
    temp4 = getMaximum(y1, y2, y3, y4);

    //corr13 = 400 - mytemp1 + mytemp3;
    //corr24 = 400 - mytemp2 + mytemp4;
    w = 100.0 * u / 50000.0;
    d1 = w * (corr13 - m + 0.001);
    d2 = w * (corr24 - m + 0.001);

    oldmin = 10000000;
    for (int i=(int) temp1; i < (int) temp2; i++) {
        for (int j=(int) temp3; j < (int) temp4; j++) {
            x = i;
            y = j;
            temp1a = fabs(sqrt(pow((x-x4),2.0) + pow((y-y4),2.0)) - sqrt(pow((x-x2),2.0) + pow((y-y2),2.0)) - d2);
            temp2a = fabs(sqrt(pow((x-x3),2.0) + pow((y-y3),2.0)) - sqrt(pow((x-x1),2.0) + pow((y-y1),2.0)) - d1);
            min = pow(temp1a, 2.0) + pow(temp2a, 2.0);
            if (min < oldmin) {
                oldmin = min;
                xSET = x;
                ySET = y;
            }
        }
    }
    //cout << xSET << "\t" << ySET << endl;

    xposition = xSET;
    yposition = ySET;


}

double getMinimum(double x1, double x2, double x3, double x4) {
    double temp;
    temp = x1;
    if (x2 < temp) {
        temp = x2;
    }
    if (x3 < temp) {
        temp = x3;
    }
    if (x4 < temp) {
        temp = x4;
    }
    return temp;
}

double getMaximum(double x1, double x2, double x3, double x4) {
    double temp;
    temp = x1;
    if (x2 > temp) {
        temp = x2;
    }
    if (x3 > temp) {
        temp = x3;
    }
    if (x4 > temp) {
        temp = x4;
    }
    return temp;
}


double getMaximum(double x1, double x2, double x3, double x4, double x5, double x6) {
    double temp;
    temp = x1;
    if (x2 > temp) {
        temp = x2;
    }
    if (x3 > temp) {
        temp = x3;
    }
    if (x4 > temp) {
        temp = x4;
    }
    if (x5 > temp) {
        temp = x5;
    }
    if (x6 > temp) {
        temp = x6;
    }

    return temp;
}

unsigned char* serialRead() {
    DWORD bytesread;
    unsigned char buffer[1];
    data[0] = 0;

    pos=0;
    while (1)
    {
        ReadFile(hCom, buffer, 1, &bytesread, NULL);
        cout.flush();
        if (bytesread)
        {
            for (int i=0; i<bytesread; i++)
            {
                data[pos++]=buffer[i];
                //cout << "read data\n";
                if (pos > 300) {
                    pos = 0;
                }

                if (pos > 5) {
                    if (!isalnum((int)data[pos-1]) && !isalnum((int)data[pos-2]) && !isalnum((int)data[pos-3])){
                        //cout << "here is " << pos << " bits of data go parse" << endl;
                        //cout << "X" << (int)data[pos-1] << "X" << (int)data[pos-2] << "X" << (int)data[pos-3] << "X"<< endl;
                        //cout << "X" << (int)data[pos-4] << "X" << (int)data[pos-5] << "X" << (int)data[pos-6] << "X"<< endl;
                        return data; // data in data, pos in pos (globals)
                    }
                }

            }
        }

    }
}



static void comSetup() {
    DCB dcb;
    DWORD dwError;
    BOOL fSuccess;

    hCom = CreateFile(serialPort,
        GENERIC_READ | GENERIC_WRITE,
        0,    /* comm devices must be opened w/exclusive-access */
        NULL, /* no security attrs */
        OPEN_EXISTING, /* comm devices must use OPEN_EXISTING */
        0,    /* not overlapped I/O */
        NULL  /* hTemplate must be NULL for comm devices */
        );

    cout << "Opened serial port " << serialPort << " " << hCom;
    if (hCom == INVALID_HANDLE_VALUE) {
        dwError = GetLastError();

        /* handle error */
        return;
    }

    /*
    * Omit the call to SetupComm to use the default queue sizes.
    * Get the current configuration.
    */

    fSuccess = GetCommState(hCom, &dcb);

    if (!fSuccess) {
        /* Handle the error. */
        return;
    }

    /* Fill in the DCB: baud=19200, 8 data bits, no parity, 1 stop bit. */

    dcb.BaudRate = 115200;
    dcb.ByteSize = 8;
    dcb.Parity = NOPARITY;
    dcb.StopBits = ONESTOPBIT;

    fSuccess = SetCommState(hCom, &dcb);

    if (!fSuccess) {
        /* Handle the error. */
        return;
    }

    COMMTIMEOUTS cto;
    cto.ReadIntervalTimeout = 4;
    cto.ReadTotalTimeoutMultiplier = 10;
    cto.ReadTotalTimeoutConstant = 100;
    cto.WriteTotalTimeoutMultiplier = 10;
    cto.WriteTotalTimeoutConstant = 100;
    if (!SetCommTimeouts(hCom, &cto)){
    }
//  UINT threadID;
//  threadHandle = (HANDLE) _beginthreadex(NULL, 0, threadMain, NULL, 0, &threadID);
}

double getDistance(double x1, double y1, double x2, double y2) {
    double temp;
    temp = sqrt(pow((x1 - x2),2) + pow((y1 - y2), 2));
    return temp;
}

double doubleabs(double x) {
    if (x < 0) {
        return x*(-1.0);
    } else {
        return x;
    }
}
```

Appendix D

Explanation of Calculations

D.1 Calculating σ

Given a set of points, the mean point, $(\bar{x}, \bar{y})$ is calculated. Next, the distance from each point to $(\bar{x}, \bar{y})$ is calculated. The standard deviation, σ, is based upon these distances.

Bibliography

[1] A. Pentland. Smart rooms. *Scientific American*, pages 274(4):68–76, April 1996.

[2] H. Ishii, C. Wisneski, J. Orbanes, B. Chun, and J. Paradiso. PingPongPlus: Design of an Athletic-Tangible Interface for Computer-Supported Cooperative Play. *Proceedings of Conference on Human Factors in Computing Systems (CHI'99)*, pages 394–401, May 1999.

[3] J.A. Paradiso, K. Hsiao, J. Strickon, J. Lifton, and A. Adler. Sensor Systems for Interactive Surfaces. *IBM Systems Journal*, 39(3 & 4):892–914, October 2000.

[4] N. Checka. A System for Tracking and Characterizing Acoustic Impacts on Large Interactive Surfaces. MS Thesis, Massechusetts Institute of Technology, Department of Electrical Engineering and Computer Science and MIT Media Lab, May 2001.

[5] Panasonic EFVRT recievers. see `http://www.maco.panasonic.co.jp/www-ctlg/ctlg/qAAA0000_E.html`.

[6] P.H. Winston. *Artificial Intelligence*. Addison Wesley, second edition, 1993.

[7] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, second edition, 1998.

[8] D.H. Johnson and D.E. Dudgeon. *Array Signal Processing*. Prentice Hall, first edition, 1993.

[9] J.A. Paradiso, C.K. Leo, N. Checka, and K. Hsiao. Passive Acoustic Sensing for Tracking Knocks Atop Large Interactive Displays. *IEEE Sensors Journal*, 17(17):1–7, June 2002.

[10] S. Haykin. *Introduction to Adaptive Filters*. Macmillan Publishing Company, 1984.

[11] H. Matthews, editor. *Surface Wave Filters*. John Wiley & Sons, 1977.

[12] D. Yelin, D. Meshulach, and Y. Silberberg. Adaptive Femtosecond Pulse Compression. *Optics Letters*, 22(23):1793–1795, December 1997.

[13] L. Cremer, M. Heckl, and R.R. Ungar. *Structure-Borne Sound*. Springer-Verlag, second edition, 1990.

[14] Saab Aerospace. The Ultimate Radar Level Gauge, February 2000.

[15] Micropower Impulse Radar. see `http://www.llnl.gov/IPandC/op96/10/10o-mic.html`.

[16] H. Ma. Ultra Wideband Impulse Radar for HCI Applications. Technical report, MIT Media Lab and University of British Columbia, June 2002.

[17] N.H. Yu.

[18] M.J. Kavaya and G.D. Emmitt. Tropospheric Wind Measurements from Space. see `http://www.ghcc.msfc.nasa.gov/sparcle/docs/MJK-GDE_invited_paper.htm`.

[19] J. Paradiso. *The Responsive Window, in* **Take Over**, *Proc. of the 2001 Ars Electronica Festival*. Springer-Verlag, September 2001.

[20] J.D. Beltran. see `http://identity.media.mit.edu/telephonestory.html`.

[21] J. Paradiso. The Interactive Window. see `http://www.siggraph.org/s2002/conference/etech/interwin.html`.

[22] M. Downey.

[23] R.A. Quinnell. Touchscreen technology improves and extends its options. *EDN*, 40(23):52, 54–6, 58, 60, 62, 64, November 1995.