A wireless positioning measurement system based on Active Sonar and Zigbee wireless nodes

CE 4710

University of Utah

19 Decemeber 2007

Christopher Jones

ketthrove@msn.com

Spencer Graff

graff_spencer@msn.com

Matthew Fisher

matthew.fisher@utah.edu

Project Website: http://radensmoke.freepgs.com/blt/

Instructor, Ken Stevens: kstevens@ece.utah.edu

Intro

3D games and animation has become one of the most popular forms of modern entertainment. Hours and hours have been spent in both making these products and enjoying them. However, controlling these 3D simulations in either development or in game play has always been a problem. High end studios can afford to use expensive motion capture systems to create somewhat realistic simulations. Even then, they are forced to hand adjust key frames to make it look correct. Home producers must start from scratch and build up.

A simpler, cheaper motion capture system would be ideal to improve on these problems. 3D animators could build miniature models that they could use to control the "bones" of their 3D figures. Game players could move their characters by moving themselves. It would even allow movement toward complete VR systems.

To move toward these goals, we developed a wireless ultrasonic positioning system. This system was designed to move objects in the scene upon the movement of ultrasonic receivers.

Hardware

Our wireless ultrasonic positioning system used the Zigbee Wireless Hardware. The project boards and the microcontrollers were from Freescale Semiconductor; they were donated by the Freescale University program and the donations were much appreciated. The hardware was obtained over the summer. The system also used basic circuit components as indicated below and special transmitter and receiver piezo-electric transducers.

Transmitter Hardware

Piezo-electric transducers from Senscomp (40KT08 and 40KR08) were used to transmit sound waves to the receivers. The overarching idea is to use the receive times of each of the four transmitters to calculate position via multilateration (see Multilateration heading of the Software section). The transmitters and receivers were set up in an active sonar mode which means that sound waves are not bounced off objects to get total distance traveled, instead, a transmitter transmits a sound wave and a receiver receives this sound wave.

These transducers can be run using a square wave with amplitude above and below a zero reference and this method is used if one does not want to worry about polarity. It was our intention to generate a square wave that stayed completely above a zero reference and so it was necessary to find the polarity of the transducers. Since there were no markings on the transducers it was necessary to hook up a voltmeter to the two leads and then direct a heat source towards the transducer. This would generate a small voltage through the pyroelectric effect and thus reveal the positive and negative terminals.

The reason we wanted to stay above a zero reference on the square wave was so that the square wave generation process would be simplified. The transducers run best at 40 kHz and this frequency was generated using the output compare function on the MCU. With the MCU we were able to generate a 40 kHz square wave with a 25 microsecond period, 50% duty cycle and a 5V amplitude. With a 5V peak the V_{rms} equals 3.53 which is not nearly enough to drive the transducers and give us the one meter minimum range. To get a higher V_{rms} it was necessary to raise the peak voltage. The maximum V_{rms} the transducers can handle is 16 which means we could go to a maximum amplitude of 22.63. We ended up using a 13.5V peak due to hardware limitations but it gave us the necessary range.

A very simple and cost effective way of generating the necessary square wave is to use an IGBT driver IC which is essentially a voltage follower. It comes in an 8 pin DIP package with two driver inputs. The square wave is supplied to the input and the driver is powered with a voltage with the desired output amplitude. Thus with a 5V square wave input you get a 13.5V square wave output. The best way to power the driver chip is with a battery and a capacitor on the supply. This generates a very "square" square wave.

Since this voltage would fry the MCU it was also necessary to put a buffer in between the driver IC and the MCU. A simple op-amp buffer with negative feedback was used to isolate the two components. This easy safety system came in very handy when a couple leads were crossed on the transmitters and 50 percent of the prototyping board was fried along with the IGBT drivers. Had the buffer not been in place our MCU would have been toast and our project would probably have had an untimely end.

Receiver Circuit

A receiver circuit was also required, which would produce pulses of proper strength to be detected by the input capture of the microcontroller whenever a 40kHz sound signal is being received through the corresponding ultrasonic receiver.

Because the amplitudes of typical signals produced by these ultrasonic receivers are very small, an amplifier and comparator are necessary.

Measurements were performed to observe the voltage strength of signals as they come directly from the ultrasonic receiver. When a ±10V, 40kHz sine wave was connected to an ultrasonic transmitter and a receiver was held directly facing the transmitter at 1m, the signal measured on an oscilloscope was a sine wave of about ±5mV.

In order to prepare a reliable signal to feed to the input capture of the microcontroller, this signal should be amplified up to the order of V rather than mV, then processed with a comparator that can be tuned to whatever threshold voltage will cause the circuit to produce pulses when a legitimate 40kHz wave is received, and reject echoes and other minor noises. Selectivity for 40kHz did not need to be implemented in the circuit because the ultrasonic transducers are reportedly already very selective for 40kHz.

A circuit was designed to amplify the ±5mV to ±1.5V—a gain of about 300. This allowed signals that are a bit weaker than the ones already observed to still produce a great enough amplitude to be reliably used by a comparator that had its own sources of inaccuracy, the most significant of which is a manually-tuned potentiometer.

The type of op-amp amplifier circuit used was a high-pass filter. This was chosen so that any DC bias would be completely rejected, which is very important because the bias may end up being even larger than the useful signal from the piezoelectric receiver. Another characteristic that was chosen for the circuit was for it to be in the non-inverting configuration—the receiver was not part of the feedback loop. Therefore the input impedance of the circuit was as high as needed, and the resonance of the crystal did not complicate the op-amp feedback.

The op-amp was chosen to have a unity-gain bandwidth much larger than the 40kHz frequency of the circuit so that the op-amp would not be a significant limitation on the quality of the signal. The op-amp also needed to have a cost that was not too high. The LM318N was chosen, which has a 15 MHz bandwidth and costs $0.45

In order to achieve a gain of 300 for 40kHz signals in a high-pass filter, the time constant of the RC feedback should be 300 times the time constant for 40kHz (4μs).

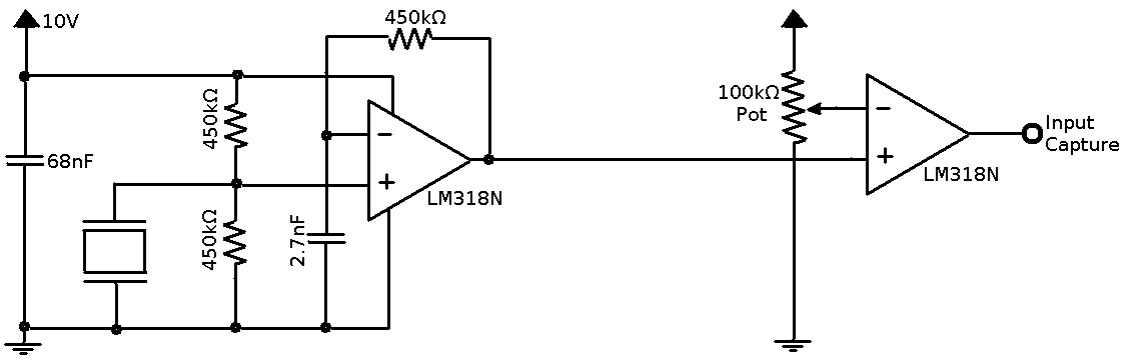
$RC = 300 * 4\mu s = 1.2$ ms

The selected component values to implement this time constant were $R = 450k\Omega$ and $C = 2.7$ nF.

On testing of the constructed amplifier circuit, the output waveform was generally extremely noisy, in spite of grounding of the breadboard and placing of capacitors across the power supplies. The solution used to handle this problem was to remove the amplifier from the breadboard, bring all the components (including the piezoelectric receiver) together into a very small volume, and put the circuit into its own small grounded package. The passive components, including a capacitor across the power supply, and the receiver were all soldered directly onto the op-amp, and the circuit was placed into a metal bottle cap (i.e. Faraday Cage). Although this solution added a third (V_{dd}) wire to the connection between the main receiver controller and each individual receiver, it brought the level of noise down so that it was generally less than the signal produced by a weak ultrasonic signal.

The comparator portion of the circuit did not need to be moved out onto the individual receiver because the usable signal was already much larger than any noise that would be acquired by the signal on its way back to the main receiver controller. The comparator simply compared the signal to a voltage from a potentiometer voltage divider, and then the resulting pulses were scaled to a safe level and fed into input captures of the microcontroller.

The final circuit constructed is shown below, with the amplifier portion on the left and the comparator portion on the right.



Software

A major component of this project was the software. The microcontroller would not function as needed, nor would the data mean much without an application to use it in. This software was divided into several components on the microcontrollers and on the host PC.

Microcontroller Software

The wireless communication software base came from Freescale. The software originally functioned as a wireless communication device between two hyper terminals on separate computers.

The base software on the MCUs had to be modified to accept the transmitter and receiver hardware and subsequently the message passing software (see the Wireless Protocol below). For the transmitters, output compares were used to synchronize when each of four transmitters began transmitting. The output compare interrupts were initialized and we had six I/O ports available for use. The Zigbee Wireless module took up pins 1 through 15 and so was not

available for use with the transmitter nodes. Initially we wanted to use more transmitters but using four gave us an accurate enough position and we were limited to adding two more anyway.

The base software on the other MCU had to be modified to accept receiver hardware and afterward the message passing software on the receiver side. We had the same I/O limitations with the receivers but we ended up using two of them and so had ample room for growth. The receivers used input capture to take their time stamps and so the same process was followed, initializing the timer modules.

Serial Interface (RS-232 and USB)

The interface from the hardware to the PC was implemented with the RS-232 serial port of the microcontroller and either a serial port or a USB port on the PC. If the USB port needed to be used, a small serial-to-USB adapter was connected to the microcontroller. The adapter would show up as a COM port on the PC when used.

A very simply serial port driver was included with the wireless UART code that was used as part of this project. The driver was set up to transmit at a baud rate of 38400.

Since the system could possibly be extended to include many ultrasonic transmitters and many ultrasonic receivers, resulting in very large amounts of data to send back to the PC, the data was sent in binary rather than text or other even more sophisticated formats.

A main goal of the serial interface is to keep functioning without crashing, as having a few erratic data points that need to be detected and thrown out is far more desirable than a system than stops functioning when there is a minor error. So the format was designed so that the leftmost two or more bits of each character transmitted over the port would indicate the use of the character data, such as a transmitter number or the high bits of a time stamp, etc.

The receiving of data into the PC was implemented using standard Win32 functions. A COM port would be opened and a new thread would be started to listen and accept each character as available from the port. The character would then be decoded according to the type of data and then placed into larger data structures.

The placing of data into the data structures is as follows: Numeric data characters would be put into the high, middle, or low bits of a number. When the character specifying the receiver number is received, the entire number would be recorded as the time stamp for that receiver. When the character indicating the transmitter number is received, all the receiver time stamps are recorded as belonging to that transmitter. When the end-of-frame character was received, a structure containing the entire frame of data was then dispatched to the multilateration/rendering program.

Wireless Protocol

In order to correctly implement the wireless system, a specific protocol was needed so that the receiver and transmitter were recording and reporting the correct data. This protocol was developed and implement in the code on the two microcontrollers.

The transmitter was started first and allowed to initialize. When the transmitter was ready, the receiver microcontroller was turned on. Upon initialization, the receiver immediately sent a start message to the transmitter, the ascii letter ‘J’, to let it know that it was ready. Upon the reception of the start message, the transmitter microcontroller sent a transmitter message, an ascii ‘T’ followed by the transmitter number (0 indexed). The receiver received the message, recorded the starting timestamp, sent an immediate acknowledge message, ascii ‘A’, then set the structure in which to record the timestamps, and enabled all the input captures for the receivers. When the transmitter received the acknowledge from the receiver it activated the first transmitter

to transmit a set number of 40kHz ultrasonic pulses. The receiver transducers received the pulses which caused the input capture to trip, a timestamp minus the start timestamp was recorded, and then the tripped input capture was disabled. After the pulses, and a delay period to allow echoes to settle, the transmitter microcontroller repeated the same process of sending the next transmitter message and activating the next transmitter. The receiver received this and again set the appropriate record structure and activated the input captures. This repeated until all the transmitters had transmitted. When each transmitter had transmitted, the transmitter sent a data request message, an ascii 'F'. When the receiver received the request message, it sent back each of the sets of recorded timestamps for each transmitter. The transmitter received the data and used the serial interface to forward the data onto the PC. After forwarding the data, it repeated the process by sending the first transmit message again.

Multilateration Software

To calculate distances from the timestamps recorded by the microcontroller we employed multilateration. Multilateration uses the differences in measured time stamps to calculate x, y, and z coordinates. To calculate three coordinates requires at least four times, and therefore in our system there needed to be at least 4 transmitters. Additional transmitters could have been added to help provide more accurate results, but we only implemented the minimum four for reasons specified above. These four time stamps are inserted into the following formulas:

$$T_L = \frac{1}{c}\left(\sqrt{(x - x_L)^2 + (y - y_L)^2 + (z - z_L)^2}\right)$$

$$T_Q = \frac{1}{c}\left(\sqrt{(x - x_Q)^2 + (y - y_Q)^2 + (z - z_Q)^2}\right)$$

$$T_R = \frac{1}{c}\left(\sqrt{(x - x_R)^2 + (y - y_R)^2 + (z - z_R)^2}\right)$$

$$T_C = \frac{1}{c}\left(\sqrt{(x - x_C)^2 + (y - y_C)^2 + (z - z_C)^2}\right)$$

Figure 1: Mulitlateration formulas

The $T_{L,Q,R,C}$ are the separate timestamps for each of the transmitters, while the $x_{L,Q,R,C,}$ $y_{L,Q,R,C,}$ and $z_{L,Q,R,C}$ are the corresponding measured positions of the transmitters. These equations are solved for x, y and z. With these four equations and four unknowns it is theoretically possible to compute the desired coordinates, but solving these equations on a computer was non-trivial and was best done with an iterative solution.

We discovered an open source GPL project that would do these calculations for us. This package, called Non-Linear Modelling and Positioning (NLMaP), has built in support for multilateration. While this package provided a way to solve the equations, it posed its own problems. First, the package would not at first compile with Microsoft Visual Studio, which the rest of the PC software for the project was using. The problem came from the packages use of non constant array initializes which the Visual Studio compiler does not support. To fix this, the code of the NLMaP package had to be edited to use dynamically allocated arrays to replace the non-compiling versions. A second problem came with ambiguous and sparse documentation on the package's multilateration classes. This made it difficult to determine what exact paramaters the functions needed in order to do the calculations. One of the last bugs of the system that was fixed was due to a misunderstanding of that the program meant by lateration measures. When the package was fixed and used correctly, however, it provided results well within our accuracy goals.

DAZ|Studio Plug-in

To test and demonstrate the positioning abilities of our wireless positioning systems we interfaced with the 3D program DAZ|Studio by DAZ3D. This program is designed for the rendering and animation of prepared 3D content. DAZ3D provides an SDK built on top of Qt framework by Trolltech. Through the use of this SDK a simple plug-in was developed for DAZ|Studio. This plug-in called the serial software to gather the time stamp data from the transmitter array microcontroller. It then scaled and filtered this data and fed it through the multilateration calculation functions. The function return values were averaged with a window of previously calculated values to help reduce jitter. This average was then applied to the object in the scene that represented the individual receiver, moving it to correspond to the measured x, y, and z values. Figure 2 shows an example of the system function and moving two dragon heads in the scene.

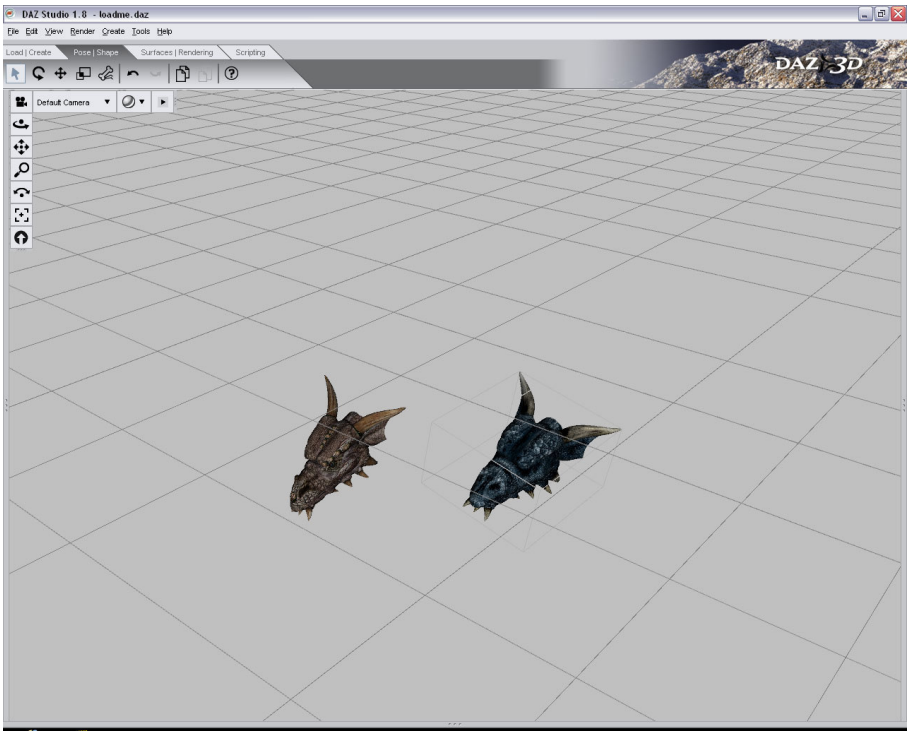


Figure 2: Screne shot from DAZ\|3D using the developed Free Form plug-in. Each dragon head corresponds to the position of a different physical receiver. Heads are from DAZ3D's Millennium Dragon 2.0.

To make the system easier to use, a simple panel was developed to quickly launch and set many of the values used by the program. Figure 3 shows a screen shot of the panel.

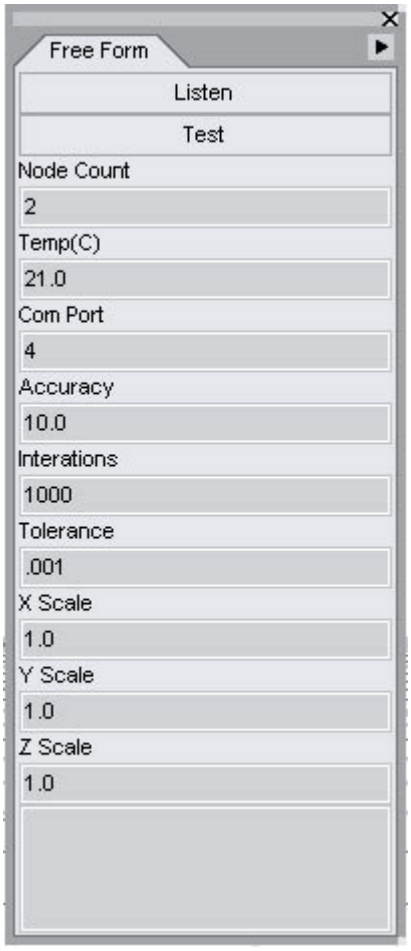


Figure 3: Free Form panel that appears in DAZ\Studio.

The listen button toggles to start/stop listening and applying the data from the transmitter microcontroller. The test button was used solely for debugging and does nothing in the final project. All the rest of the boxes, but the last, allow the labeled setting to be set. Most are self explanatory. Node count determines how many nodes to insert and apply transforms to; this should match the receiver count. The temperature should be set the current temperature in the room in Celsius. The temperature was used to calculate the speed of sound in the room. The COM port box should be set to the COM port the transmitter serial connection is assigned to. Accuracy is used to set at what error level to throw out received results. Iterations and tolerance

are used by the multilateration component to calculate the results. The most accurate results are achieved with small tolerance values (<.01) and large iterations values (>100), though this takes more time. At a certain point, the calculations reach a level of accuracy and changing the numbers will have no effect. The last 3 boxes scale the results along the indicated axes.

The DAZ|Studio plug-in as a whole was implemented in a straight forward fashion. Most of the problems encountered in it were the result of the process of learning to use the SDK and its Qt framework. However, by using the SDK, the plug-in can be simply distributed to other Windows machines with DAZ|Studio program to allow them to interface with the wireless ultrasonic positioning system.

Conclusion

The project was a success. The positions of the ultra sonic receivers were accurately displayed in the DAZ|Studio interface. Further development could allow the use of the system to move figure bones instead of just simple objects. It could also be extended to allow more receivers to animate more complicated models.

References

In bibli:

[1] CSM12C32
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=HCS12C32SLK&fsrch=1

[2] MC13213
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC13213&fsrch=1

[3] MCU PROJECT BOARD-2
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=PBMCUSLK&parentCode=HCS12C32SLK&nodeId=0162469544#

[4] IEEE 802.15.4
http://www.ieee802.org/15/pub/TG4.html

[5] MC13192U
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=ZIGBEESLK&fsrch=1

[6] 40KT08
http://www.senscomp.com/specs/40KT08%20%20spec.pdf

[7] 40KR08
http://www.senscomp.com/specs/40KT08%20%20spec.pdf

[8] Ultrasonic Distance Measurement with the MSP430
http://www.focus.ti.com/lit/an/slaa136a/slaa136a.pdf

[10] Multilateration on Wikipedia
http://en.wikipedia.org/wiki/Multilateration

[11] NLMAP
http://www.cl.cam.ac.uk/research/dtg/research/wiki/NLMAP

[12] DAZ3D
http://www.daz3d.com

[9] Caffeine is a xanthine alkaloid compound and a very useful molecule
http://en.wikipedia.org/wiki/Caffeine