LEGO NXT ACOUSTIC TAPE MEASURE APPLICATION

A CAPSTONE SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTERS OF SCIENCE

IN

INFORMATION
&
COMPUTER SCIENCES

MAY 2011

By

成玉
Cheng, Yu

Supervisor:
Edoardo S. Biagioni, PhD

1 Abstract

The Acoustic Tape Measure (Distance) program is free and open-source software distributed with the XO-1 computer, which was developed as part of the One Laptop Per Child (OLPC) project. The Distance program determines the physical distance between two XOs by measuring how long it takes sound pulses to travel between them. [1]

The Lego Mindstorms NXT is a programmable robotics kit released by the Lego Group. [2] A variety of development strategies exist for the NXT, [3] and this project used leJOS NXJ, which is a Java-based replacement firmware for the NXT microcontroller. leJOS is a tiny Java Virtual Machine that provides a subset of the standard Java Runtime Environment (JRE) as well as a small set of classes specifically designed to work with the NXT sensors and motors.

In this project, we studied the algorithm of the XO Distance application. We explored the hardware constraints and possibilities of implementing the Distance algorithm on the Lego NXT. We then implemented this algorithm as an autonomous program running on two NXTs.

Contents

List of Figures

List of Tables

2 Introduction

2.1 OLPC

OLPC is a project aiming to create educational opportunities for the world's poorest children by providing each child with a rugged, low-cost, low-power, connected laptop with content and software designed for collaborative, joyful, self-empowered learning. It is believed that the educational situation can be immediately improved by transferring the responsibility of learning directly to the children. When children are given access to the world's information they may learn in collaboration with one another. Thus, the need to rely on a complex infrastructure, or teachers to obtain an education is eliminated.[4] The OLPC project's current focus is on the development, construction, and deployment of the XO-1 laptop and its successors.

2.2 XO

The OLPC XO computer, shown in Figure 1, is an inexpensive sub-notebook computer. The XO was designed as a learning tool and built especially for children in developing countries, living in some of the most remote environments. [5] It is about the size of a small textbook. It has built-in Wi-Fi and Bluetooth and a unique screen that is readable under direct sunlight for children who go to school outdoors. It is durable, functional, and energy-efficient. [6]

Figure 1: XO Computer

The OLPC project with its XO-1 computers is regared as one of the major factors that led top computer hardware manufacturers to begin creating low-cost laptop computers, known as Netbooks, for the consumer markets. [7] Soon after the introduction of the OLPC, developing countries were given a much larger choice of Netbook vendors, such as Dell, Acer, and HP.

XOs run the Sugar Learning Platform, and their applications are called Sugar Activities. There are about 160 Activities available for XOs and 14 of them are pre-installed. Available XO Activities have a wide range of usage from education, social networking, and entertainment.[8] Some examples are shown in Table 1.

Activity Name	*Activity Description*
BROWSE	Web browser
DISTANCE	Measure distance between XOs
PIPPY	Python Programming
RECORD	Video and audio capture
TERMINAL	Activity version of the Sugar terminal
WRITE	Word processor

Table 1: Sugar Activity Examples

2.3 Acoustic Tape Measure (Distance) Program

The Distance Activity is a pre-installed XO Activity. It measures physical distances using sound pulses. Two XO computers are needed to use this activity. Both XOs must be connected to the same shared network. When the activity is started on one computer, it sends invitations to all neighboring computers. Once the second computer accepts the invitation, both XOs start to provide an option to begin measuring distances. Figure 2 shows the activity screen when it's in action.

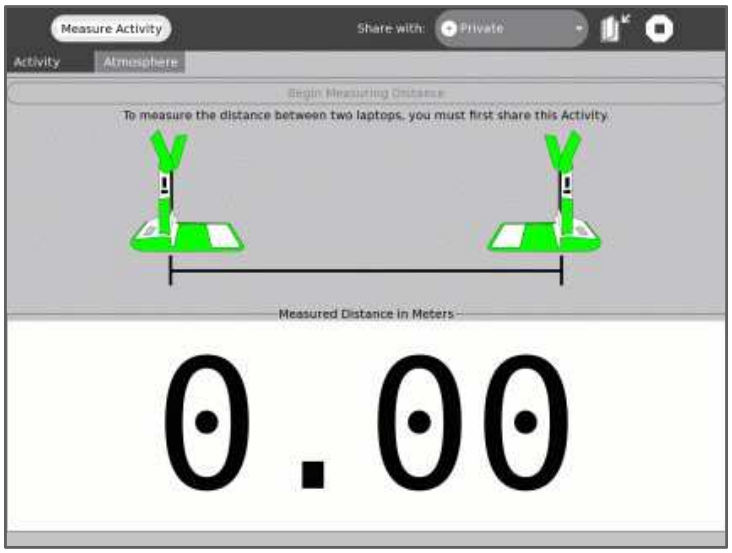

Figure 2: Distance Activity in Action

The application performs well within a range of approximately 10 meters. When the XOs do not face each other, however, the measurements are inaccurate. In this case, both XOs display synchronized readings,

those readings fluctuate greatly and are inaccurate. Figure 3 shows the incorrect and correct ways of using the Distance Activity.

Figure 3: Distance Activity with Incorrect Placement (left) and Correct Placement (right)

2.4 Lego Mindstorms

Lego Mindstorms, shown in Figure 4, is a line of programmable robotics/construction toys, manufactured by the Lego Group. It comes in a kit containing many pieces including sensors and cables. Mindstorms originated from the programmable sensor blocks used in the Lego line of educational toys. [9] This project used Lego Mindstorms NXT 2.0, which was released in 2009.

Figure 4: Lego NXT with Sensors

2.4.1 Lego NXT

The main component in the Lego Mindstorms kit is a brick-shaped computer called the NXT Intelligent Brick. It can take input from up to four sensors and control up to three motors via RJ12 cables. The

brick has a 100×64-pixel monochrome LCD display and four buttons that can be used to navigate a user interface using hierarchical menus. It also has a speaker and can play sound files at sampling rates up to 8 kHz. Power is supplied by 6 AA batteries (1.5 V each) in the consumer version of the kit, and by a Li-Ion rechargeable battery and a charger in the educational version. [11] The specs for the CPU, RAM, hard disk drive, and sensors are listed in Table 2.

NXT Brick		***NXT Sensors***	
CPU	Pentium II	Touch	0 or 1
RAM	32 MB	Ultrasonic	0 ~ 170 (cm)
HDD	115 MB	Light	Intensity 0 ~ 100
		Color	RGB 0 ~ 255
		Sound	Intensity 0 ~ 100
		Rotation	0 ~ 360 (degrees)

Table 2: NXT Specs

2.4.2 NXT Programming

The NXT-G toolkit is a programming package that distributed with Lego Mindstorms. It features an interactive drag-and-drop environment, and it is useful for writing simple programs. NXT-G is powered by LabVIEW, a popular programming environment created by National Instruments. LabVIEW uses data flow programming to create a virtual instrument. [10] Figure 5 shows an example program using NXT-G. In this program, we used only the light sensor. This program runs in a loop and plays a sound when the reading from a light sensor crosses a threshold value.

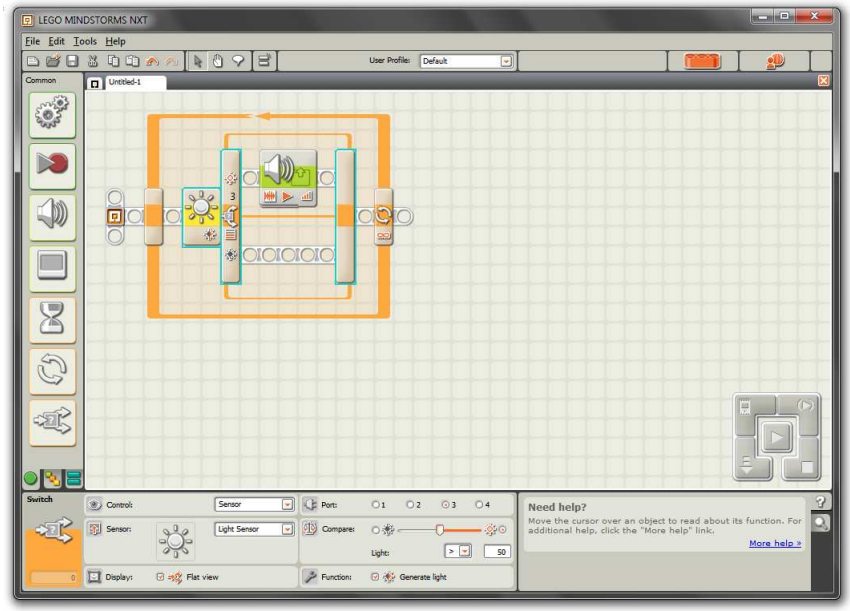

Figure 5: NXT-G Example Program

The NXT-G software is adequate for basic programming on the robot, such as driving motors, incorporating sensor inputs, doing calculations, and learning simple programming structures and flow control. [12] A more advanced programming language is required for the purposes of this project. Many options were availabe but leJOS NXJ was chosen for this project because it is a high-level, open-source language based on Java. leJOS application execute on the NXT using a custom firmware that provides a subset of the typical JRE. [3]

3 Related Work

3.1 Acoustic Tape Measure Algorithm

The Acoustic Tape Measure algorithm on the XO relies on the ability of the XO microphone to hear the XO speaker. The algorithm works by turning on the microphones of both XOs and then having both XOs play distinctive sounds. One XO hears its own sound almost immediately, but there is a propagation delay before it hears the sound from the other XO. The XO that plays second hears the first XO's sound delayed and hears its own sound immediately. Therefore, the first XO measures a longer interval between the two sounds, and the second XO measures a shorter interval. The difference between these measurements is twice the propagation delay, and combining this information with the approximated speed of sound, the XOs can determine their distance apart. [1]

Acoustic techniques similar to this have been used successfully in other applications. For example, acoustic localization based on common time difference of arrival (TDOA) has been used in outdoor wireless sensor networks (WSNs). [13, 14, 15, 16].

A visual explanation of the Distance algorithm is given below. Figure 6 shows the interaction of the activities with respect to time.

Symbol	***Description***
M_1	The interval between recording s1 and s2 on computer #1.
M_2	The interval between recording s1 and s2 on computer #2.
t_p	The propagation delay due to the distance of the two computers.
t_1	The interval between playing a sound and hearing the sound on the same computer.
t_2	The interval between hearing a sound and recording the sound.
t_3	The interval between recording a sound and playing a sound on computer #2.

Table 3: Description for Symbols used in the Algorithm

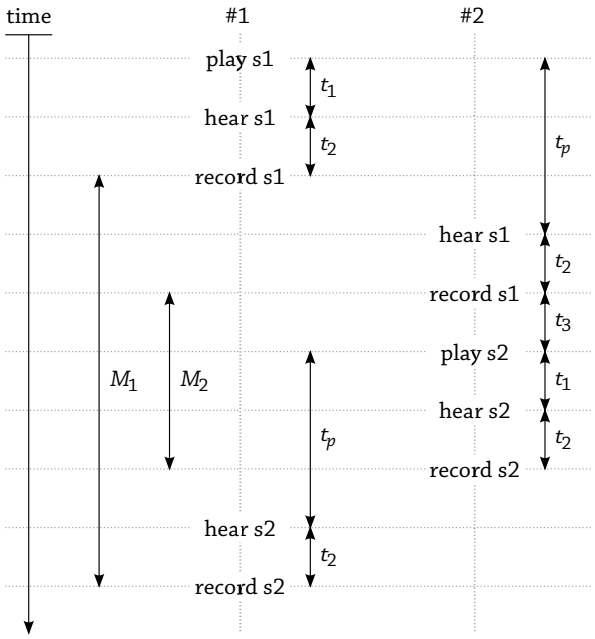

Figure 6: Acoustic Tape Measure Algorithm

The relevant time intervals with respect to M_1 are shown in red in the left diagram in Figure 7. The relevant time intervals with respect to M_2 are shown in green in the right diagram of Figure 7.

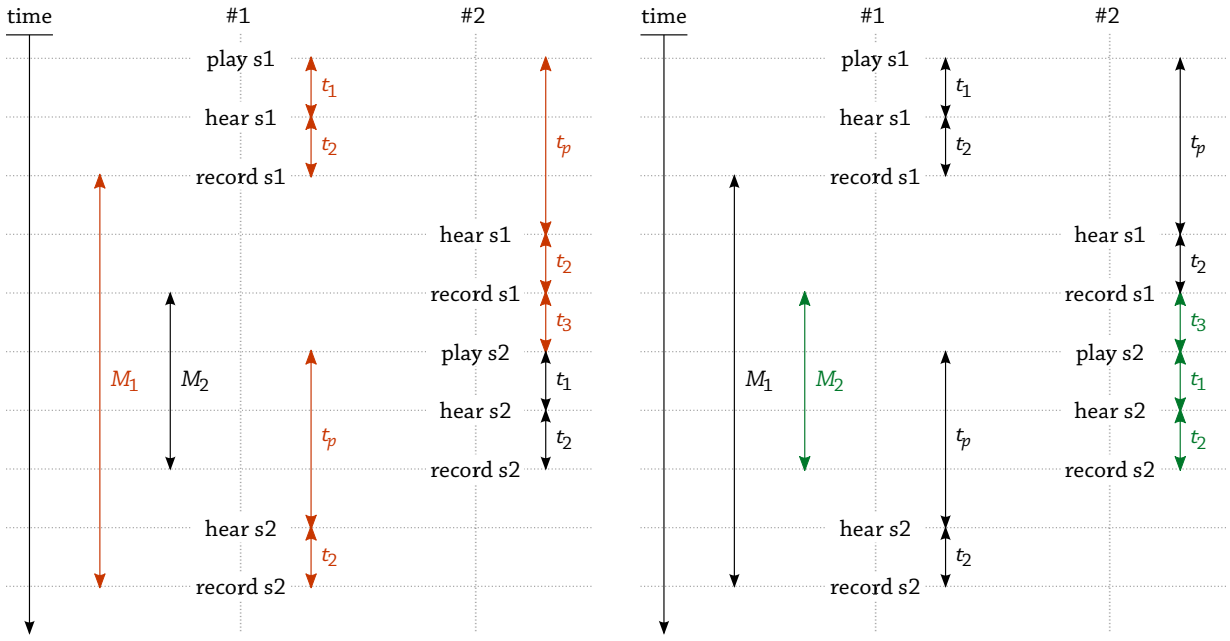

Figure 7: Timelines Relevant to M_1 (left) and M_2 (right)

From the diagram, it is clear we can express M_1 with t_p, t_1, t_2, and t_3.

$$\begin{aligned} M_1 &= t_p - (t_1 + t_2) + t_2 + t_3 + t_p + t_2 \\ &= 2 \cdot t_p + t_3 + t_2 - t_1. \end{aligned}$$

Similarly, we can express M_2 with t_1, t_2, and t_3.

$$M_2 = t_3 + t_1 + t_2.$$

With this information, we can express the time difference of M_1 and M_2 with the other terms. The time intervals of sounds are the measurable variables in this situation, and we will use them to calculate the physical distance of two XOs.

$$M_1 = 2 \cdot t_p + t_3 + t_2 - t_1$$

$$M_2 = t_3 + t_1 + t_2$$

$$\Rightarrow M_1 - M_2 = 2 \cdot t_p - 2 \cdot t_1.$$

Since we assumed that an XO hears its own sound almost immediately, we can approximate t_1 as zero and rewrite the equation as below, where d denotes the distance and c_{sound} denotes the speed of sound, which in 80 °F dry air is 347 m/s.

$$M_1 - M_2 = 2 \cdot t_p - 2 \cdot t_1$$

$$d = t_p \cdot c_{\text{sound}}$$

$$\Rightarrow d = \frac{M_1 - M_2 + 2 \cdot t_1}{2} \cdot c_{\text{sound}}$$

$$\because t_1 \approx 0$$

$$\Rightarrow d \approx \frac{M_1 - M_2}{2} \cdot c_{\text{sound}}.$$

Note that for this calculation to hold, we need to assume three environmental and hardware conditions. First, we need to assume c_{sound} is the same at the two computers. Second, we need to assume the sound sensor of a computer is sufficiently close to its speaker. In this case, t_1, the time interval between playing a sound and hearing a sound on one computer, can be approximated to zero. Last but not least, we need to assume that t_2's are the same on both computers. This condition can be satisfied only when the two computers have exactly the same hardware, and hence, we can guarantee the identical time interval

between hearing a sound and recording the sound by implementing the sound recording module the same way on the two computers.

In other words, this algorithm does not work if the two computers are located at two places where the speeds of sound are different, e.g., due to significantly different temperatures or air pressures; The algorithm does not work if the speaker and sound sensor on one machine are too far a part to omit the time interval of sound traveling from the speaker to the sensor on one machine; The algorithm does not work if the two computers have different hardware or different instruction sets to record a sound on the client and server machines.

3.2 NXT Java Setup

3.2.1 leJOS NXJ Firmware Setup

Setting up the NXT brick to run Java involves installing Java on the PC, installing the NXT USB driver on the PC, and installing leJOS on the PC and on the NXT brick. The NXT-G operating system is replaced with leJOS, a Java-based firmware, and the NXT brick then carries a limited version of the JRE and JVM.

3.2.2 Embedded Programming Environment

In this project, we used the Eclipse Integrated Development Environment (IDE). External tools were used for compiling the code into a leJOS project and downloading the program into the NXT brick. These tools were bundled with the firmware by the leJOS team. Figure 8 shows options in Eclipse to link and upload a leJOS application to the NXT. This program displays “Hello, world” on NXT’s LCD until a button is clicked.

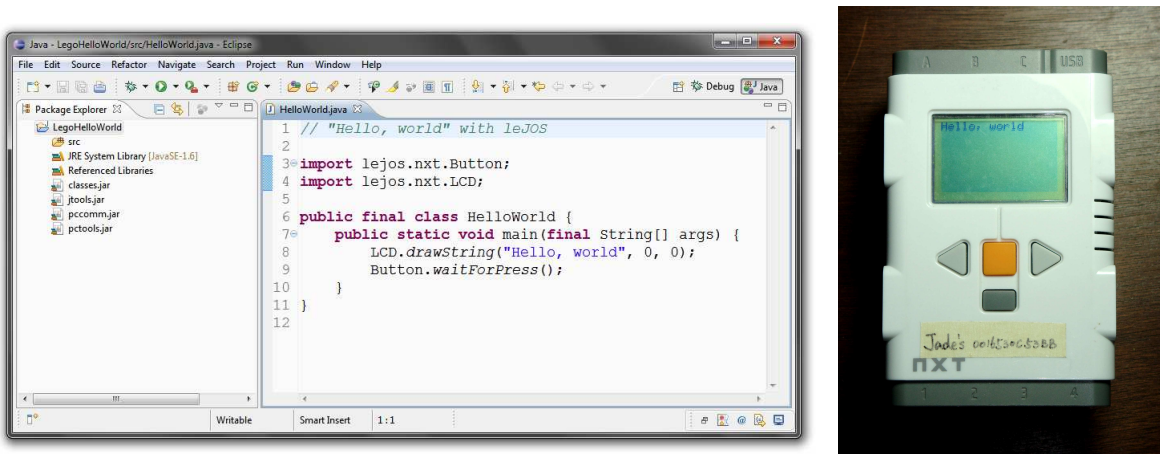

Figure 8: Eclipse for leJOS

3.2.3 Runtime Diagnostics

The speed of the NXT LCD is inadequate for diagnosing and debugging high-speed operations in real-time. As an alternative, the leJOS provides a remote console module that is intended to display LCD output as well as debugging messages from running applications. Figure 9 demonstrates the code to start the remote console as well as the console output for the HelloWorld program.

Use of the remote console requires a connection, either USB or Bluetooth. Bluetooth is preferable, but the instability in the wireless connection can at times limit the usefulness of this debugging technique.

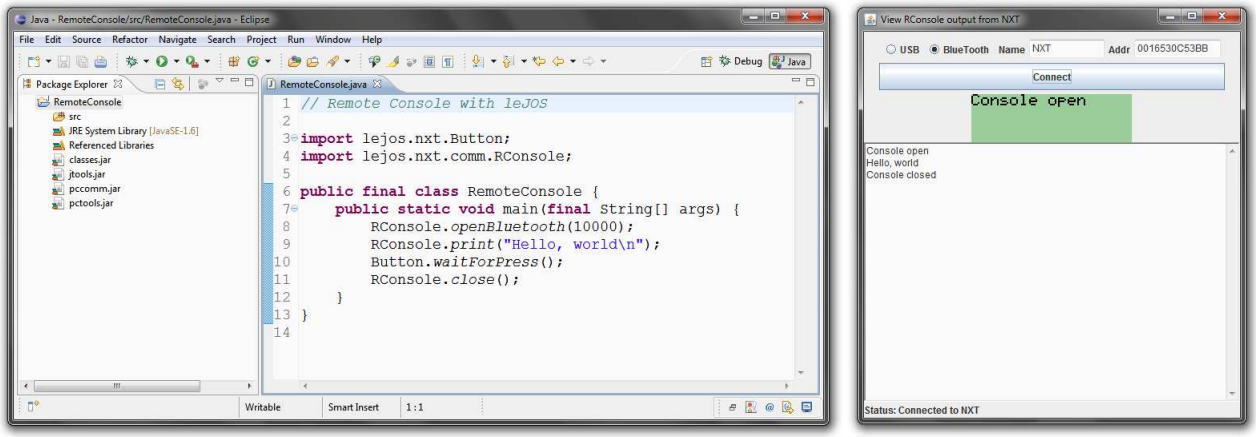

Figure 9: leJOS Remote Console

3.3 NXT Sound System

We explored the possibility of implementing the XO Acoustic Tape Measure algorithm on the Lego NXT brick. To ensure the Distance algorithm would generate reasonable measurements, the NXT hardware needed to meet some basic requirements shown in Table 4.

Component	***Requirement***
Speakers	To play sounds louder than ambient noise
Sound Sensors	To detect sounds from an NXT speaker
System	To record and play sounds simultaneously

Table 4: Basic Hardware Requirement for the Distance Implementation

To use the algorithm with three or more participants and to make measurements simultaneously, the NXT would need two additional capabilities, which are shown in Table 5. If the hardware could not meet these requirements, the participants would be required to follow a pre-defined protocol and take turns emitting sounds.

Component	*Requirement*
Speakers	To play distinctive sounds louder than ambient noise.
Sound Sensors	To detect distinctive sounds from NXT speakers

Table 5: Basic Hardware Requirement for Multiple Participants

3.3.1 Speakers

The Lego NXT speakers are capable of playing a specified frequency (Hz) at a specified volume (% of the system sound) for a specified duration (ms).

3.3.1.1 Frequency

In the first experiment, we tested the NXT's ability to play different frequencies at a fixed volume. We programmed the NXT to play at 200 Hz, 400 Hz, 600 Hz, 800 Hz, and 1000 Hz. All sounds were played at 100% volume for a duration of 1 second on and 2 seconds off. The results were as expected.

3.3.1.2 Volume

In the second experiment, we tested the NXT's ability to play different volumes at a fixed frequency. We programmed the NXT to play at 20%, 40%, 60%, 80%, and 100% of its maximum volume. All sounds were played at 800 Hz for a duration of 1 second on and 2 seconds off. The results were as expected.

Although the NXT speakers were not particularly loud, from these preliminary tests, it seemed the NXT speakers would be adequate for the Distance application in reasonably quiet environments.

3.3.2 Sound Sensor

The Lego NXT sound sensor detects sounds in two modes, decibels (DB) mode and adjusted decibel (DBA) mode. In DB mode, all sound frequencies are measured with equal sensitivity, and the sound sensor is capable of detecting some sounds that are too high or too low for the human ear to hear. In DBA mode, however, the sensitivity of the sensor is adapted to the sensitivity of the human ear. In other words, the sensor attempts to ignore sounds that humans are unable to hear.[17]

The use of DB and DBA modes to detect ranges of frequencies has been studied previously[17]. In this project, however, it was desirable to know whether or not the capabilities of the sound sensor would be adequate to distinguish frequencies generated by the NXT speaker.

Figure 10: NXT Sound Sensor

3.3.2.1 Frequency and Mode

In this experiment, we programmed one Lego NXT brick as a sound generator that played tones at a range of frequencies. we programmed another Lego NXT brick as a sound receiver that recorded all sounds it heard and printed those recorded values to the LCD in real time. The generator and receiver were placed within one inch of each other.

Figure 11 shows the recorded values per frequency for DB and DBA modes.

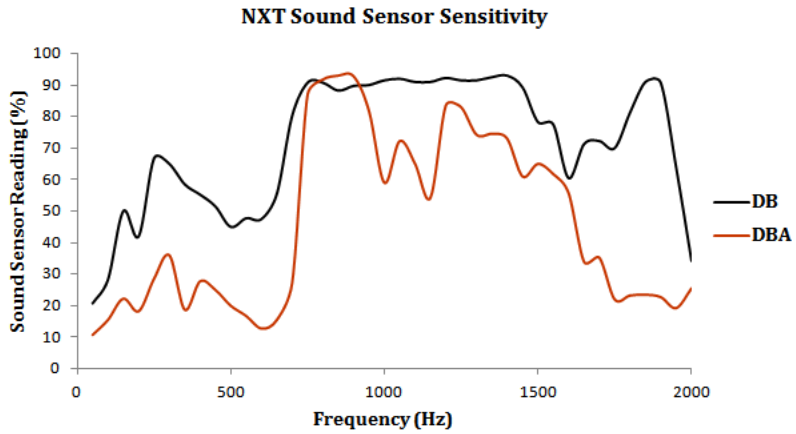

Figure 11: NXT DB mode vs. DBA mode

This experiment demonstrated that the NXT sound sensor is able to distinguish sounds produced by the speaker of another NXT. Both DB and DBA modes were adequate, although the largest, most stable range of distinguishable frequencies were in DB mode between 800 and 1400 Hz.

3.3.2.2 Sound Pulse Detection

In this experiment, we programmed one Lego NXT brick as a sound generator that played a fixed frequency, which was determined as the most sensitive frequency from the previous experiment, 1100 Hz.

We programmed another Lego NXT brick as a sound receiver that recorded all sounds it heard and transmitted those recorded values to the PC. The generator and receiver were placed within one inch of each other.

Figure 12 shows the recorded values for DB mode. Sounds were played at 100% volume for 0.5 seconds on and 0.5 seconds off.

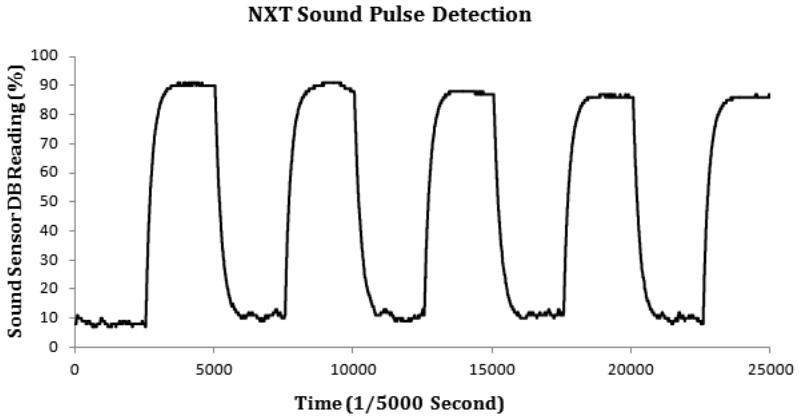

Figure 12: NXT Sound Pulse

This experiment demonstrated that the NXT sound sensor does not record sound as frequencies (oscillations). Even though it can capture approximately 5000 samples per second, the frequency of the sound is lost, and only the amplitude (i.e., the envelope) is reported. In other words the raw data captured by the sound sensor is insufficient to distinguish sounds of different frequencies. Other successful applications using similar acoustic techniques have relied on a sensor's capability of detecting frequencies of sounds. [13]

This experiment also demonstrated that the NXT is easily able to detect pulses of sound when they are louder than ambient noise. The levels reported by the sound sensor rose quickly and consistently after the generation of the tone.

3.3.2.3 Distance Sensitivity

In this experiment, two Lego NXT bricks were programmed in a similar way as in the previous experiment, and the receiver printed the recorded values to the LCD in real time.

Figure 13 shows the recorded values in DB mode. Sounds were played at 100% volume at a distance ranging from 0.5 inches to 38 inches.

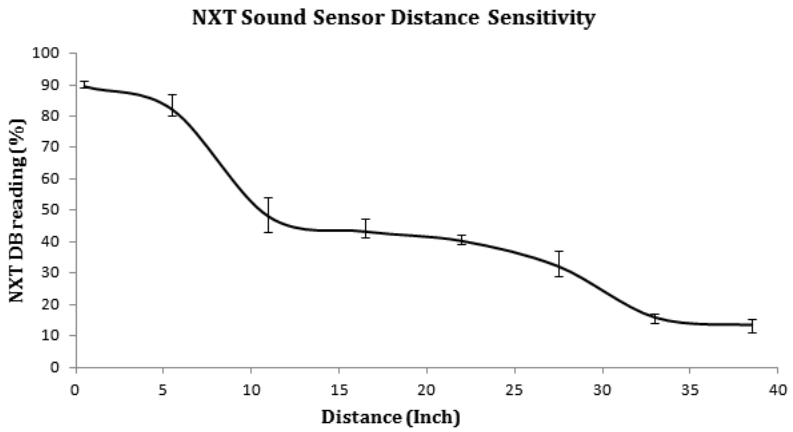

Figure 13: NXT Sound Detection at Distances

This experiment demonstrated that the NXT sound sensor can distinguish sound from another NXT within only 30 inches (0.7 meters). This greatly reduces the usefulness of implementing the Distance algorithm on the Lego NXT, especially considering the following two observations. One, the accuracy of the algorithm is greatly diminished when the sender and receiver are in close proximity. Two, the sensible range for distance measuring is less than the claimed capability range of the sound sensor.

3.3.2.4 Echo Influence

During the aforementioned experiments, we noticed that echo greatly affected the sound measurements. When the sound receiver was located in the corner of a room, and the sound generator was located away from the corner but facing it, the receiver recorded DB readings at approximately 25% higher than it did when the two NXTs were placed in an open area. This is likely due to sound reflections from the walls.

4 NXT Distance Implementation

4.1 State Machine Design

We implemented the following state machines for the client unit (initiator) and the server unit (responder). The application starts by allowing the user to specify the NXT's mode of operation, client or server, by pressing the left or right arrow buttons. The two NXTs then execute their state machines and continuously display synchronized distance results until the user presses the escape button.

4.1.1 Client State Machine

After the client initiates the connection with the server, it enters its state machine, which is shown in Figure 14. It starts by playing a tone (play s1) and immediately recording the tone (record s1). After waiting for the tone to finish playing (stop s1), it waits for a tone from the server (record s2). After it receives the second tone, it sends the timing M_1 to the server (send M_1) and then receives the distance d from the server (receive d). It displays the distance and then waits a while (stop s2) before playing another sound (play s1).

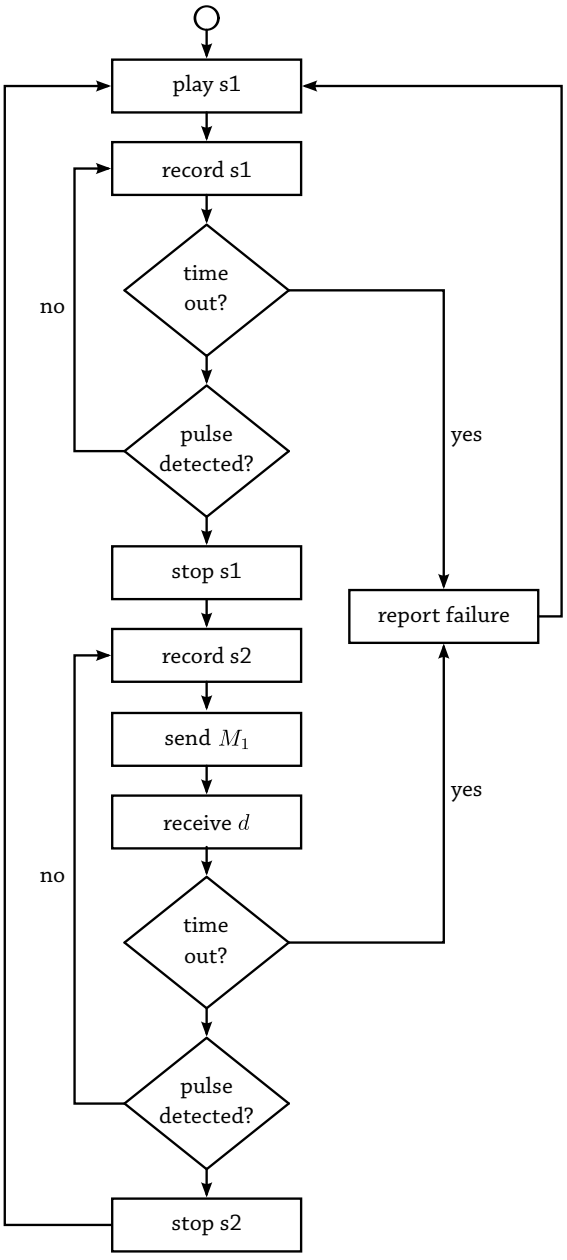

Figure 14: Client State Machine

4.1.2 Server State Machine

After the server accepts the connection from the client, it enters its state machine, which is shown in Figure 15. It starts by recording a tone played by the client (record s1). After waiting for the tone to finish playing (stop s1), it plays a tone (play s2) and immediately records this tone (record s2). After receiving the timing M_1 from the client (receive M_1), it calculates and displays the distance d, and then it sends the calculation to the client (send d). It waits a while (stop s2) before recording another sound (record s1) .

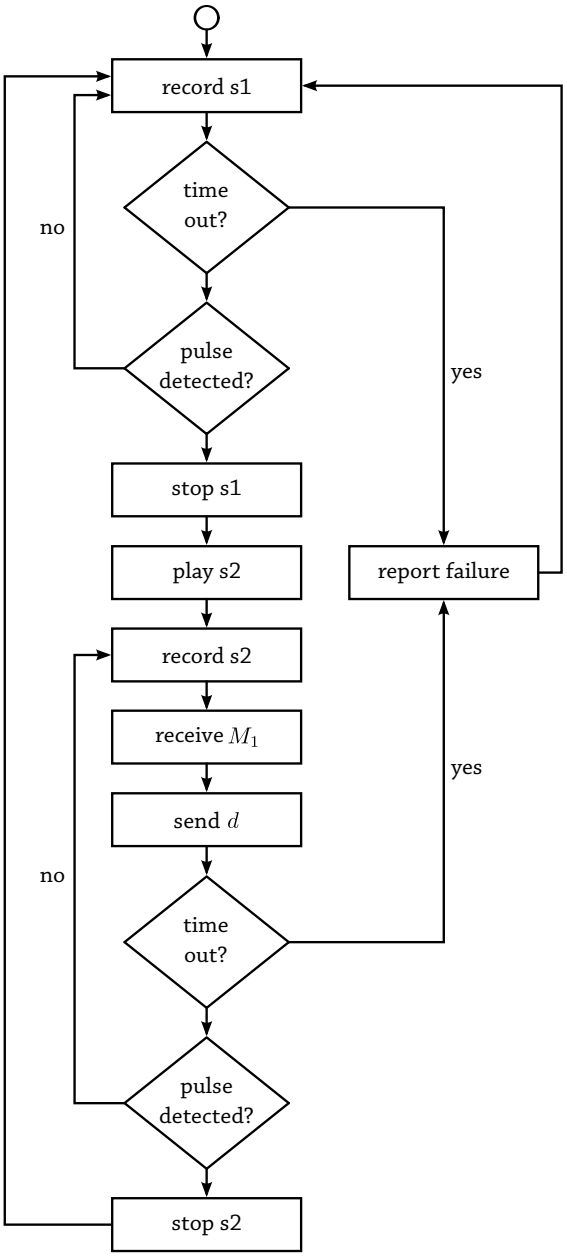

Figure 15: Server State Machine

4.2 Event Timeline

In an ideal scenario, the timeline of events would occur as shown in Figure 6. In real applications, however, there is a need to handle various errors. For instance, the program needs to recover from undetected recordings.

Figure 16 shows the timeline for ideal cases in which when both parties detect sounds from each other. There are two complete cycles shown in the diagram, where p represents the time between two consecutive sound pulses. This time corresponds to the interval between the rising edges of the pulses, and it is several times longer than then length of an individual pulse.

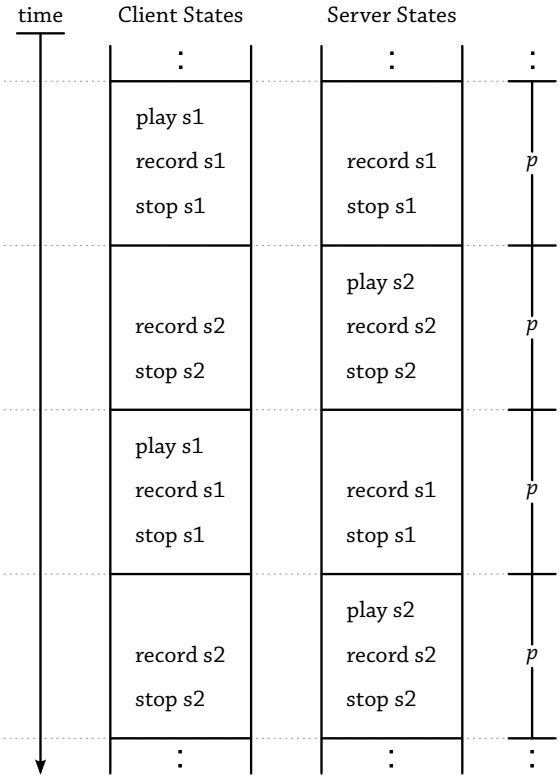

Figure 16: Event Timeline in an Ideal Scenario

The application enters a failure state any time it fails to detect an expected sound pulse. It recovers from this condition, displays a failure message, and then enters the next measurement cycle normally. Figure 17 shows the timeline when this occurs.

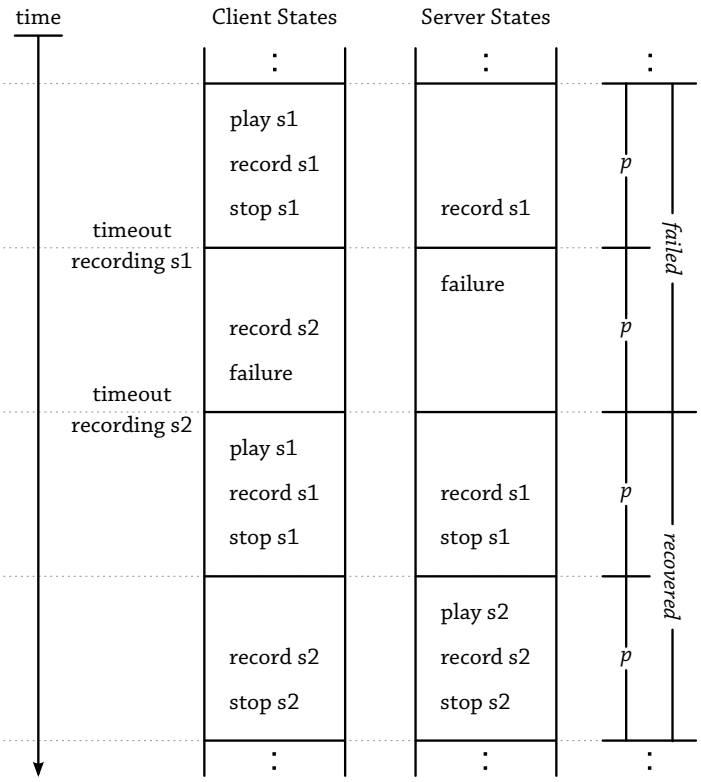

Figure 17: Event Timeline with an Occurrence of Error

4.3 Program in Action

Both the client and server NXTs run the same version of the program, and the user must select which state machine they execute. After the client initiates the connection, the two NXTs take turns playing tones, and they display synchronized measurements. The program terminates when the user presses the escape button. If only one participant is left running, it will continue to function, but it will continuously display failures.

Just like the XO computers, it is best to position the NXTs so their speakers and microphones face each other. Otherwise the distance measurements are inaccurate. The program is capable of recovering from errors. When the NXT fails to detect the other participant, it displays a failure and continues to function. When the sound from the other participant becomes detectable, the NXT recovers from the error and continues to report measured distances.

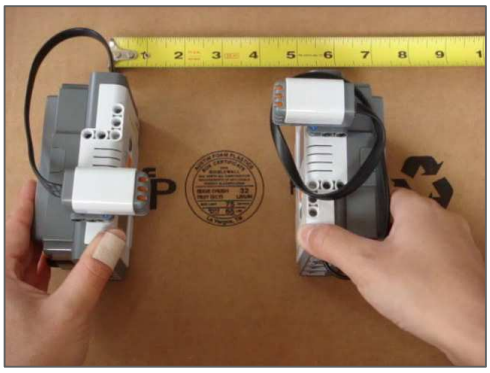

Figure 18: NXT Distance Program in Action

4.4 Program Results

As discussed in Section 3.1 Distance Algorithm and Section 3.3 NXT Sound System, there are some hardware limitations. In order the generate consistent readings, the NXTs must be in a quiet environment, away from walls, without obstacles between them, and without physical disturbances.

Recordings were computed using the following formula, where c_{sound} is 1.366×10^{-5} inches per nanosecond, corresponding to 347 meters per second, which is the speed of sound in 80 °F.

$$d \approx \frac{M_1 - M_2}{2} \cdot c_{\text{sound}}.$$

Figure 19 shows recorded distance versus actual distance for 6 different distances: 2 inches, 4 inches, 6 inches, 8 inches, 10 inches, and 12 inches. For each distance, 25 continuous recordings were used as data for the chart

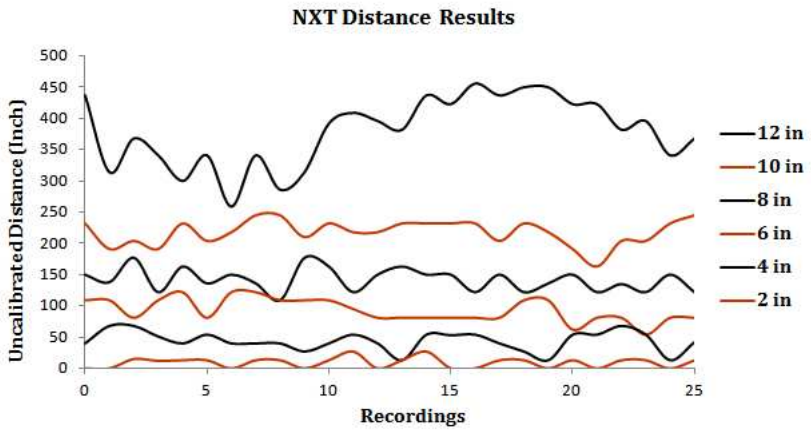

Figure 19: NXT Distance Results

Figure 20 shows the average recorded distance versus actual distance.

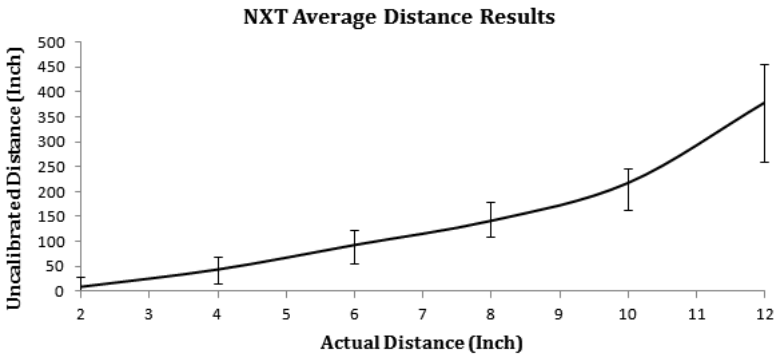

Figure 20: NXT Average Distance Results

5 NXT Distance Analysis and Discussions

The NXTs do not calculate the correct distances, but the values they calculate are consistent and do directly correspond to the actual distances of separation, i.e., by introducing a constant scaling factor such as 20.

Recall the two assumptions of the Distance algorithm: t_1 approximates zero, and the four durations of t_2 in the event timeline are the same. The implementation must satisfy these conditions. On the NXT the speaker and sound sensor are in close proximity, so basic physics tells us that t_1 is approximately zero. To satisfy the second condition, we carefully constructed the code so that t_2 was calculated using identical bits of code, i.e. a single method. In other words, all four different sound recording states of the two different state machines recorded sounds in identical ways.

In spite of this, there is clearly a discrepancy between the actual distances and the recorded distances. To investigate this problem, we performed two additional experiments on the NXTs.

5.1 Attack Time Analysis

5.1.1 Attack Time vs. Pulse Durations

Figure 21 shows the results of our first experiment. We programmed an NXT as a sound generator and receiver. We had it play a fixed tone at different durations: 1/2 second, 1/20 second, and 1/200 second.

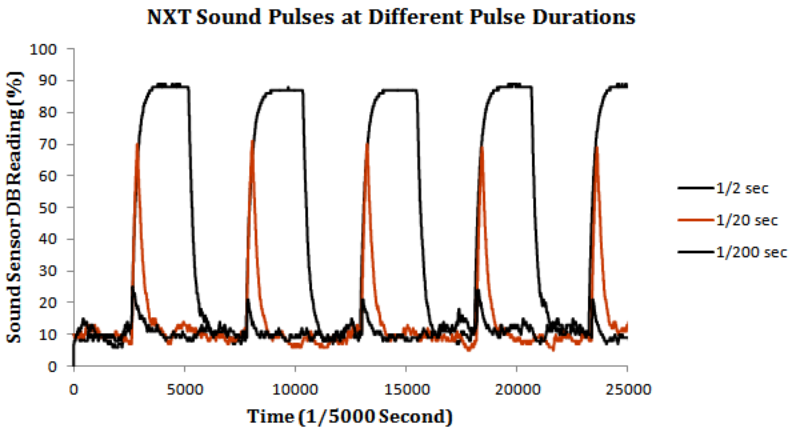

Figure 21: NXT Sound Pulses at Different Pulse Durations

From this experiment, we learned that short sound durations would not suffice for the NXT Distance program. This is due to the fact that the attack time of the sound pulses is long, and a short duration, such as 1/20 second (50 milliseconds), is not enough time for the sensor to detect a tone generated by another NXT at its peak volume.

5.1.2 Attack Time vs. Distances

For relatively long attack times, the time to reach the user-defined DB threshold seemed inconsistent. This threshold was set to be 30% DB value for the NXT Distance executions in Section 4.4, Program Results. This effect could greatly influence the distance measurement. To study this further, we performed a second experiment. We programmed one NXT as a sound pulse generator and a second NXT as a sound receiver. We played a fixed tone at a fixed duration at different distances: 2 inches, 6 inches, 10 inches, and 12 inches. Figure 22 shows the results of this experiment.

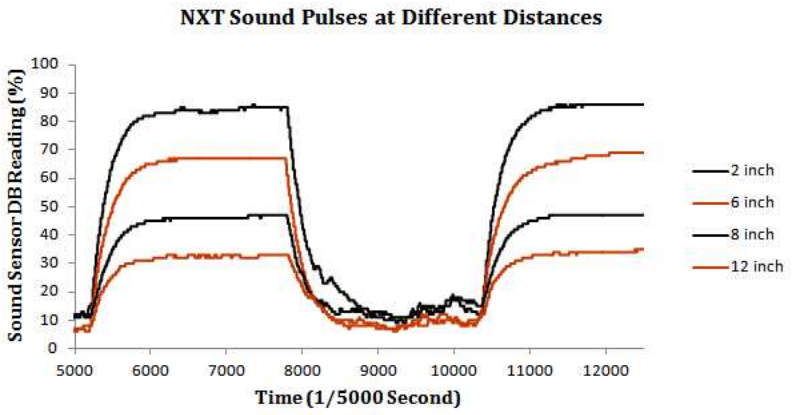

Figure 22: NXT Sound Pulses at Different Distances

This experiment demonstrated that it would take significantly different periods of time to reach the user-defined DB threshold, based solely on the distance of separation between the two NXTs. Figure 23 demonstrates this situation.

Symbol	***Description***
t_1	The amount of time to reach the DB threshold for a sound pulse generated at a close distance.
t_2	The amount of time to reach the DB threshold for a sound pulse generated at an average distance.
t_3	The amount of time to reach the DB threshold for a sound pulse generated at a far distance.

Table 6: Description for Symbols used in the Attack Time Comparison Chart

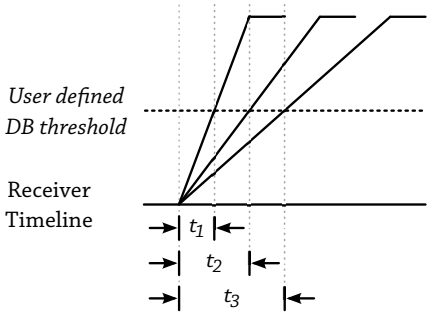

Figure 23: NXT Attack Time Comparison

5.2 Attack Time Influence in the Distance Algorithm

What we have found is the interval between two sound pulses measurements, $M_1 - M_2$, is in fact greater than $2 \cdot t_p$, which was the original conclusion. The new formula is shown below, where t_d is delay caused by different attack times.

$$M_1 - M_2 \quad = \quad 2 \cdot t_p + 2 \cdot t_d.$$

Figure 24 demonstrates the origin of this new term:

Symbol	***Description***
M_1	The interval between recording s1 and s2 on the client
M_2	The interval between recording s1 and s2 on server.
t_p	The propagation delay due to the distance between the two computers.
t_{d_1}	The delay caused by different attack times at distance d_1.
t_{d_2}	The delay caused by different attack times at distance d_2.

Table 7: Description for Symbols used in the Attack Time Effect Chart

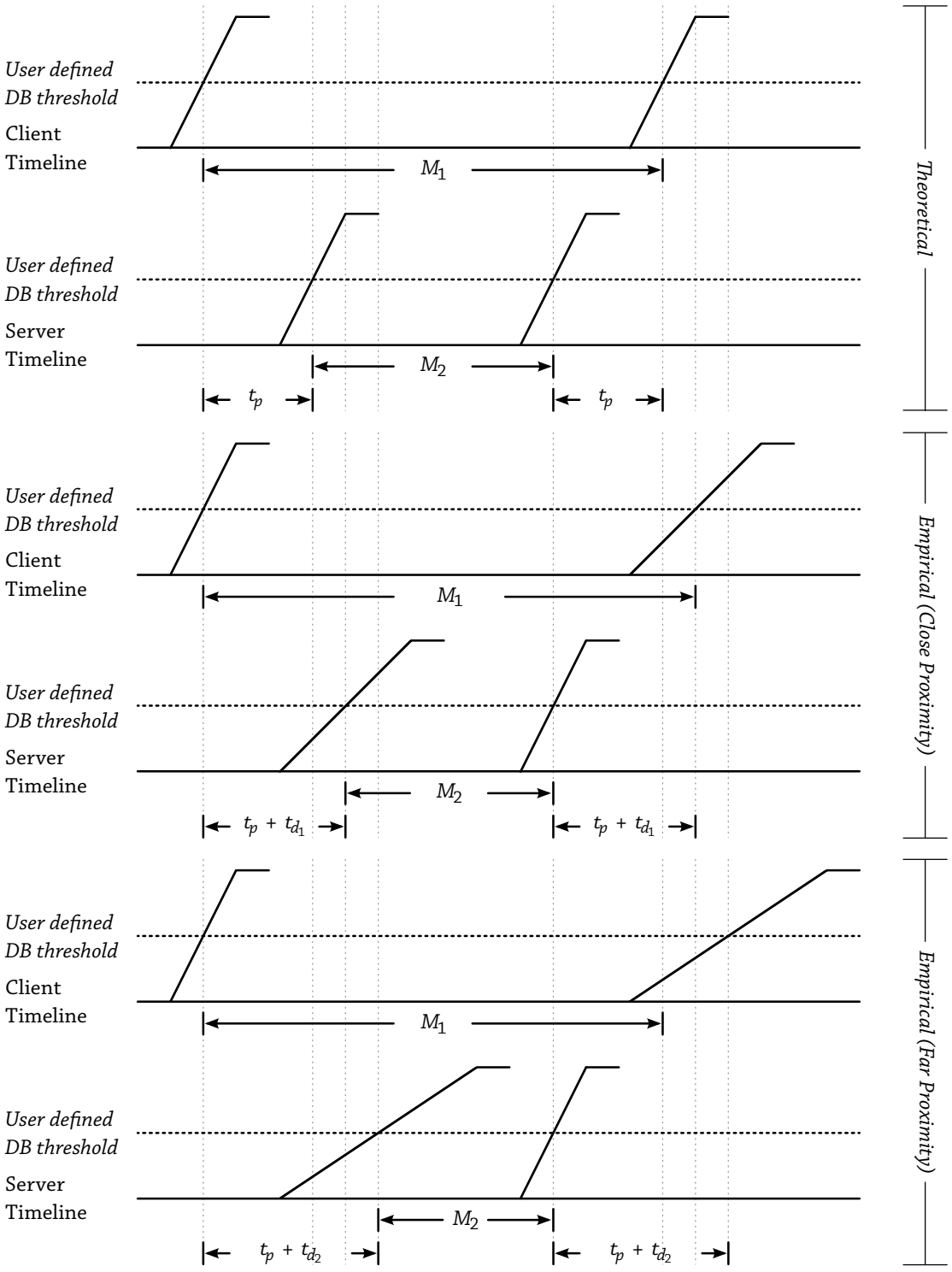

Figure 24: NXT Attack Time Effect

As discussed previously, the usefulness of this algorithm is mainly restricted by the quick diminishing of sound sensitivity as the distance increases. NXT sound sensors are incapable of detecting sounds from other NXTs that are located only a couple of feet away. In the experiments above, the maximum distance

shown was 12 inches, which is about as far away as it is possible to record accurate distances. Extremely quiet environments would allow the NXTs to distinguish sounds with greater sensitivity and the measurements within a detectable range can easily be calibrated so that the NXT reports an correct distance recording. But due to capability of the NXT speaker and sound sensor, the usage of this application would still be limited.

6 Conclusions

In this project, we studied the algorithm of the Acoustic Tape Measure (Distance) program, which is distributed with the XO computer from the OLPC project. We explored the Java programming environment of the Lego NXT from the Lego Mindstorms robotics kit. After performing a series of experiments with test programs, we implemented the Distance algorithm as an autonomous program running on two NXTs. We then analyzed the application performance and noticed a consistent error term. Inspired by these results, we conducted more experiments and derived the origin of this error, the relatively slow attack time of the speaker. For this reason, sounds originating at different distances were detected with substantially different attack times, and this breaks the Distance algorithm by introducing an un-modeled time delay. To investigate this further, we should be able to implement the algorithm on other hardware without much modification to the state machines introduced by this paper.

In this project, we explored an alternative NXT firmware and SDK called leJOS. The leJOS SDK utilizes a standard Java compiler to build images that execute on the NXT. The JRE is limited but does support network communications, threading, and a variety of classes to control the NXT hardware. Overall, combined with Eclipse, leJOS and the Lego NXT are proved to be an educational toy and could be a good candidate for introductory computer science and software engineering courses. [18]

References

[1] Acoustic Tape Measure. http://wiki.laptop.org/go/Acoustic_Tape_Measure

[2] Lego Group (January 4, 2006). "What's NXT? LEGO Group Unveils LEGO MINDSTORMS NXT Robotics Toolset at Consumer Electronics Show". Press release. Retrieved 2007-09-17.

[3] leJOS, Java for Lego Mindstorms. http://lejos.sourceforge.net/index.php

[4] Bentley, C. (2007). "The OLPC Laptop: Educational Revolution or Devolution?" in Proc. 2007 World Conference on ELearning in Corporate, Government, Healthcare and Higher Education. Quebec, Canada, pp. 647-652.

[5] Ward, Mark (September 27, 2007). "BBC NEWS - Technology - Portables to power PC industry". BBC News. Retrieved 2008-01-25.

[6] OLPC XO. http://laptop.org/en/laptop/

[7] Netbook History. http://en.wikipedia.org/wiki/Netbook#History

[8] XO Activities. http://wiki.laptop.org/go/Activities

[9] Lego Mindstorms. http://en.wikipedia.org/wiki/Lego_Mindstorms

[10] LabVIEW. http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT_2.0#NXT-G

[11] Lego NXT. http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT#NXT_Intelligent_Brick

[12] NXT-G. http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT#NXT-G

[13] Jingbin Zhang, Ting Yan, John A. Stankovi, Sang H. Son (2007). "Thunder: towards practical, zero cost acoustic localization for outdoor wireless sensor networks". ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 11, No.1, pp.15–28.

[14] Y. Kwon, K. Mechitov, S. Sundresh, W. Kim, G. Agha (2005). "Resilient Localization for Sensor Networks in Outdoor Environments", in Proceedings of ICDCS, 2005.

[15] J. Sallai, G. Balogh, M. Maroti, A. Ledeczi, B. Kusy (2004). "Acoustic Ranging in Resource-Constrained Sensor Networks", Technical Re- port, ISIS-04-504.

[16] A. Savvides, C.C. Han, M.B. Srivastava (2001) "Dynamic fine-grained lo- calization in ad-hoc wireless sensor networks", in Proceedings of MobiCom, 2001.

[17] DB-DBA Study. http://www.convict.lu/htm/rob/NXT_sound_sensor.htm

[18] Michael W. Lew, Thomas B. Horton, Mark S (2010). "Using LEGO MINDSTORMS NXT and LEJOS in an advanced software engineering course," in Proc. 2010 23rd IEEE Conference on Software Engineering Education and Training (CSEET), IEEE Computer Society. Washington, DC, pp.121-128.