ROOM RECONSTRUCTION AND NAVIGATION USING ACOUSTICALLY OBTAINED ROOM IMPULSE RESPONSES AND A MOBILE ROBOT PLATFORM

BY

CHRISTOPHER THOMAS CO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Professor Mark Hasegawa-Johnson

ABSTRACT

This work explores the design and effectiveness of a robot that uses a combination of active sonar and a pseudo-random acoustic signal to navigate and reconstruct an unknown environment. The robot sends the pseudo-random signal into the environment and records the resulting response. This response is processed to gain information on the robot's immediate surroundings. Previous work done in this area focused on the recording and processing aspect to determine the location of a sound source or multiple sound sources. We apply similar algorithms to localize what are known as virtual sound sources. Virtual sound sources are created when sound from a sound source reflects off of a surface, such as a wall or object, and are recorded by a receiver. The recorded reflected sound is commonly known as an echo. A virtual sound source is placed at the location where the sound incident to the receiver would have originated from had no reflection taken place. By placing the real sound source and receivers on the same platform, if we can accurately localize the generated virtual sound sources, we can compute the path the sound wave took and localize the surfaces off of which the sound wave reflected. We remember these surface locations to generate a map for the robot to use when navigating through the unknown environment. Finally, we present a method to improve the accuracy of the baseline virtual sound source localization algorithm by using quadratic interpolation.

To my parents, for their love and support.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Professor Mark Hasegawa-Johnson. Aside from the vast technical knowledge he provided, he also challenged me to pursue new areas and ideas, and was a great source of inspiration and advice when I faced obstacles both in this work and in my life. Without his guidance and unending patience, none of this would be possible.

In addition, I want to extend my thanks to Lae-Hoon Kim. Lae-Hoon first proposed the idea to use a tetrahedral microphone array on a mobile platform to find walls and objects in a room. He was also a great resource in the early stages of this work.

Finally, I would like to thank all of my friends and colleagues both in undergraduate and graduate studies. They made my time here very enjoyable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IR	Impulse response
RIR	Room impulse response
MLS	Maximum length sequence
RMS	Root mean square
ISM	Image-source method
D/A	Digital-to-analog
A/D	Analog-to-digital
DOA	Direction of arrival
TDOA	Time delay of arrival
GCC	Generalized cross correlation
SLAM	Simultaneous localization and mapping

CHAPTER 1

INTRODUCTION

In this work, we study the idea of reconstructing the physical shape of the surrounding environment solely through the use of sound and investigate how this reconstruction applies to robot mapping and navigation. In this introductory chapter, we describe the motivation behind this work as well as previous work done in this area. Finally, we outline the remainder of this document.

1.1 Motivation

Humans use two main senses to perceive their surroundings: sight and hearing. We focus our research on the sense of hearing. Our sense of hearing can be used to help judge the general location of an object that creates a sound, known as a sound source. If you were to close your eyes and focus on the sounds you hear, you could discern the general direction and location of each sound source in your immediate area. This is because the sound source emits a sound wave that eventually reaches your ears (a type of sound receiver), and our brain processes what we hear to determine the origin of the sound. This is the basic idea behind sound localization. Sound localization can give the listener information one would not normally be able to obtain in a visually noisy environment.

The process of using sound localization to obtain information on the surrounding environment for navigation use is commonly known as sonar, which is short for sound navigation and ranging. There are two types of sonar: active sonar and passive sonar. Passive sonar involves localizing surrounding objects that emit a sound. Active sonar involves emitting pulses of sounds from a known location and listening for the sound pulse reflection off of the surface of an object through receivers also in known locations. The distance

to the object is determined from the time delay between the sound source emission and reception as well as the speed of sound in the medium the sound traveled though. We focus on active sonar techniques for our robot, but we acknowledge that efforts in passive sonar can be used to augment our robot's picture of its immediate environment.

One popular type of active sonar frequently studied is echolocation or biosonar. Echolocation is used by animals with poor sight such as bats and dolphins. These animals are able to navigate complex areas with only the use of a sound source (mouth) and two receivers (ears). These animals emit a sound and process the resulting sound reflections, called echoes, to determine the distances to nearby objects. Plenty of research has gone into deciphering how these animals process echoes [1, 2, 3].

We will apply this active sonar idea to a robotic platform in an effort to produce a robot that can navigate through a visibly poor area. The robotic platform will contain a sound source and multiple sound receivers, all placed in a specific configuration to best obtain measurements of the surrounding environment. We apply source localization algorithms to localize what are known as virtual sound sources. Virtual sound sources are created when sound from a sound source reflects off of a surface, such as a wall or object, and are recorded by a receiver. The method places a virtual source at a location beyond the reflective surface such that the amplitude, timing, and direction of arrival of the reflection would be duplicated, if the reflective surface were not present, by direct sound from the virtual source. By placing the real sound source and receivers on the same platform, if we can accurately localize the generated virtual sound sources, we can compute the path the sound wave took and localize the surfaces the sound wave reflected off of. We remember these surface locations to generate a map for the robot to use when navigating through the unknown environment.

Autonomous navigation of a hazardous or complex area is a primary goal of robotics because the robot could be sent into a dangerous situation instead of risking a human life. Aside from the ability to operate in visibly poor areas, environments reconstructed from acoustics can be sampled with less information compared with vision. This is due to the sparse nature of acoustical reflections. With sound, when the propagating sound wave hits an object, its echo will only contain information about the closest surface orthogonal to the sound source/receiver combination. In the end, the intention

is not to replace visual navigation with acoustic navigation but to show just how one can augment the perception of one's environment through the use of acoustics.

1.2 Related Work

A lot of robots today use ultrasonic sensors to detect walls directly in front of the sensor. Ultrasound refers to sound with frequencies above 20 kHz, which humans cannot naturally hear. In Kurz [4], a robot was outfitted with 24 ultrasonic range sensors in order to learn and navigate from a starting point to defined end point. In Ohya et al. [5], a rotating ultrasonic sensor was used to find the normals to the surrounding walls and construct a map using this information.

We focus our efforts on audible sound, which has a frequency range of 20 Hz to 20 kHz. One reason for this choice is because this range is the natural frequency range humans hear, so this work will investigate further into how humans perceive their environment. In addition, work with audible sound could carry potential benefits for robots in urban environments. Currently, robots that use ultrasonic sensors must have both an ultrasonic emitter and receiver. Working with audible sound, which is very prevalent in environments with humans, could lead to audible sound generators, like human speech, to help augment the map constructed by the robot — passive sonar. A better map could be constructed due to the abundance of audible sound generators.

Sound source localization is a heavily researched topic. In Atmoko et al. [6], an algorithm was developed which used three or more microphones in a linear microphone array to find a sound source in 3D space. In Valin et al. [7] and Hu et al. [8], the robots used in each work required eight microphones to localize a sound source in 3D space. In Xu et al. [9], a robot with a tetrahedral array, like the one we will use, was able to localize an external sound source. While sound source localization is prominent in acoustics literature, environment reconstruction using sound localization techniques is sparse. Furthermore, there has been even less research done with robots taking acoustically reconstructed maps and using them to navigate. Gunel [10] developed a robot that used a rotating acoustic sensing apparatus to obtain

directional impulse response measurements which were used to reconstruct size and shape of the room, however, no navigation work was done.

For the rest of this work, when we discuss sound, we are specifically talking about sound in the audible frequency range.

1.3 Organization

We targeted our work toward reconstructing enclosed indoor spaces or rooms. As such, we investigate room impulse responses, room reconstruction methods, and rectangular room simulations. We begin this thesis with a background review of acoustics, room reconstruction methods, and basic navigation algorithms in Chapter 2. In Chapter 3, we formally propose the overall robot system design that emits a sound and records the resulting echos which are used to construct a map of the room from the robot's view from which the robot can use to navigate. In addition, we formulate methods to increase the accuracy of current room reconstruction methods in this chapter. We present the results of our simulations in Chapter 4, and in Chapter 5 we discuss the implications of our results and conclude with remarks of our contributions and potential future work.

CHAPTER 2

BACKGROUND

In order to understand the reconstruction method, we first introduce the concept of a room impulse response and its relationship with sound wave propagation.

2.1 Room Impulse Response

The impulse response (IR) is the output of a system when presented with an impulse as an input to the system. One such system could be a room. Information about the physical characteristics of a room can be derived from the room's impulse response (RIR). Rooms can be assumed to be a linear time-invariant system given that there are no moving objects in the room. The input to the system is a sound played inside the room from a sound source and the output of the system is the recorded sound at a sound receiver.

2.1.1 Sound Propagation

Sound travels through a medium as a wave with a specific frequency and amplitude. Many techniques have been made to model the propagation of sound [11]. There are wave-based, ray-based, and statistical modeling methods. In this work, we focus on ray-based modeling and in particular the ray-trace representation of sound propagation. Figure 2.1 shows this ray-trace representation as sound travels through a room from a sound source to a sound receiver. Ray-trace representation tracks the wavefront of the sound and isolates specific sound paths the sound wave traveled along. So, although sound travels in waves, since we are only interested in the path of the wavefront of a sound originating from a source to a receiver, this ray-trace representation is preferred.

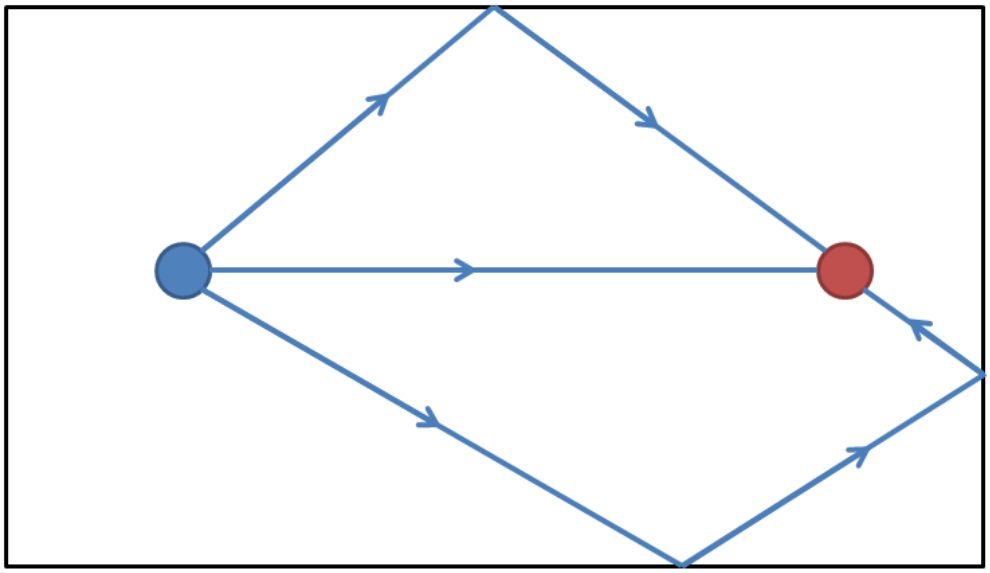

Figure 2.1: Ray-trace representation of sound paths traveling in a room from sound source (blue) to receiver (red).

2.1.2 Correlation of Sound Propagation to Room Impulse Response

The ideal room impulse response consists of impulses which denote the arrival of the sound from the sound source to the receiver. A sample ideal room impulse response is shown in Fig 2.2. In a measured room impulse response, the spikes or peaks in the impulse response represent the impulses in the ideal impulse response.

Room impulse responses consist of three components — direct sound, early reflections, and late reverberation. Direct sound is a single peak that corresponds to the shortest travel path between sound source and receiver. Early reflections are the set of discrete reflections whose individual reflections can be separated or perceived. Early reflections are further separated into two parts: first reflections and second/other reflections. First reflections pertain to the first peaks seen after the direct sound and correspond to a single sound reflection off a wall or surface. Second/other reflections correspond to the later reflections after the first reflection but are still clearly distinguishable. Finally, late reverberation, also sometimes called the reverberation tail, contains so many reflections that they are no longer able to be separated.

From the room impulse response, we can derive the orientation and dis-

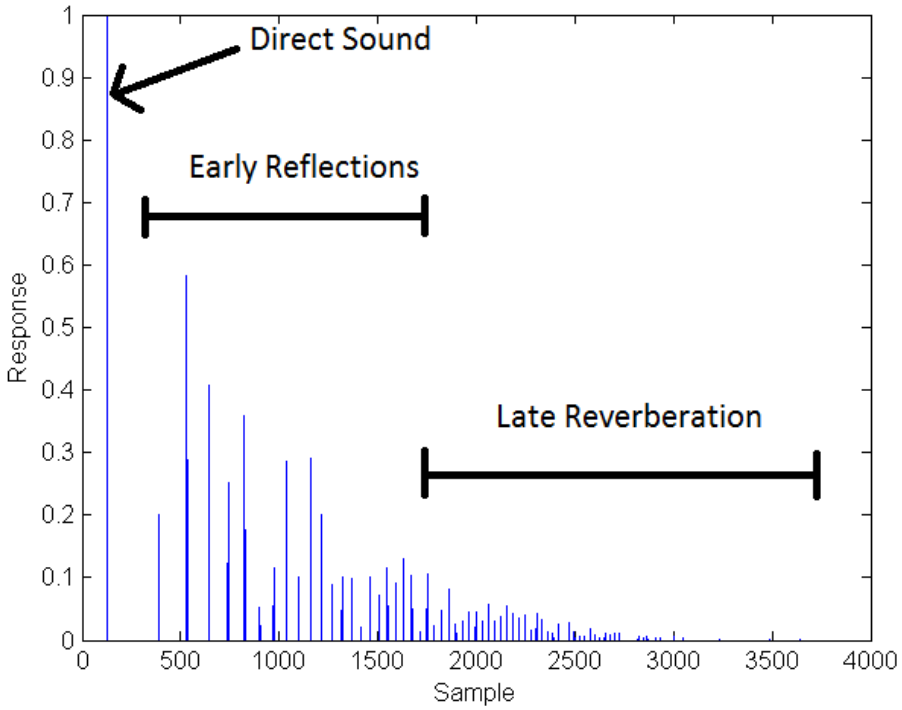

Figure 2.2: Ideal room impulse response for a 4 m by 4 m by 4 m room where the sound source is located at (3 m, 2 m, 2 m) and receiver at (2 m, 2 m, 2 m).

tance of the walls in relation to the listener. Early reflections represent sound bouncing off a nearby object or wall before being recorded by the listener. If we know the direction a reflection came from, the distance the sound signal has traveled, and the location of the sound source and receiver, we can accurately predict the point(s) of reflection.

Looking closer at Fig. 2.2, the arrival of sound between a sound source and receiver is a single impulse function in the ideal RIR. Each impulse function is characterized by:

$$h(t) = a\delta(t - \tau) \tag{2.1}$$

where δ is the unit impulse function, a is an amplitude coefficient, and τ is the arrival time of the sound. The ideal RIR is a summation of these impulse functions.

In an actual system, the sound source input, $x[n]$, must first undergo digital-to-analog (D/A) conversion to produce $x(t)$ which is convolved with $h(t)$ to produce room response $y(t)$. $y(t)$ passes through an anti-aliasing

low-pass filter and is sampled to produce $y[n]$. The anti-aliasing filter assumed in this thesis is an ideal lowpass filter:

$$g(t) = sinc(\omega_c t) \tag{2.2}$$

where ω_c, the cutoff frequency of the lowpass filter, is set to $\frac{3}{5}\pi$ as a simplified model of a non-ideal A/D.

Each of the samples, n, corresponding to a peak in the sampled RIR is characterized by

$$s(t) = h(t) * g(t) = sinc(\omega_c(t - \tau)) \tag{2.3}$$

$$s[n] = s(nT) \tag{2.4}$$

where T is the sampling period. The sampled RIR is the summation of $s[n]$ for all n samples of the RIR.

2.2 Maximum Length Sequence

A lot of work has gone into creating different methods to estimate or measure a room's impulse response. The applications of the methods range from echo cancellation to fabrication of a virtual acoustic room [12, 13]. The most common technique involves using an impulse-like sound, such as a gunshot or pop of a balloon, and recording the resulting measured response. This approach is the most direct method as the resulting measured response ideally will be the room's true impulse response. However, due to the physical nature of gunshots or balloon pops, the measured responses come close but are not exactly the ideal impulses responses. The fact that these inputs are not ideal impulses leads to non-linearities in the resulting response and thus the resulting measured response may not accurately represent the room's true impulse response.

There are other methods that use specific input signals and postprocessing of the resulting measured response to extract the room impulse response. The two most common methods for input signal are pseudo-random white noise and time-varying frequency signals [14].

We chose to use the maximum length sequence (MLS) technique to obtain

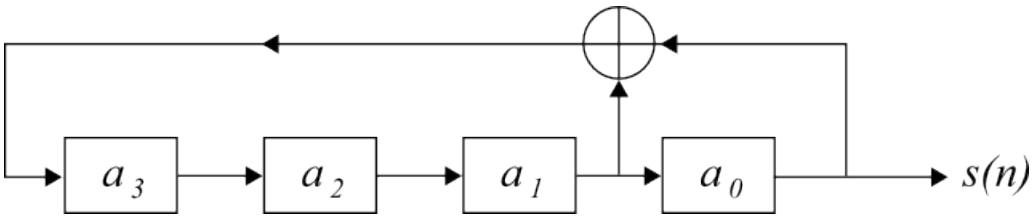

Figure 2.3: Example of MLS generation using a linear feedback shift register of length 4.

the room impulse response. The MLS technique was first introduced by Schroeder in 1965 [15] and is based on exciting the room with a periodic pseudo-random signal known as a maximum length sequence. This maximum length sequence is produced using linear feedback shift registers. The order of the MLS is determined by the shift register length (m) so the resulting sequence length generated is $N = 2^m - 1$. Figure 2.3 shows an example of a linear feedback shift register of length 4. For a linear feedback shift register system with length m, the bit values inside the registers can be computed using:

$$a_k[n+1] = \begin{cases} a_0[n] \oplus a_1[n] & k = m-1 \\ a_{k+1}[n] & otherwise \end{cases} \tag{2.5}$$

where n is the time index, k is the bit register position, and $\oplus$ represents addition modulo 2.

The circular autocorrelation of an MLS yields a 1 at time zero and $-1/N$ elsewhere so as N increases, the autocorrelation approaches the unit impulse function. This property gives the MLS signal similar stochastic properties to white noise, which is essential to the room impulse response reconstruction process.

To obtain the room's impulse response using the MLS technique, an analog version of a predetermined MLS signal is applied to a room and the resulting response is measured [16, 17]. The impulse response is obtained by performing a circular cross-correlation with the original sequence which produces a periodic impulse response $h'[n]$ which is related to the linear impulse response $h[n]$ by the following equation:

$$h'[n] = \sum_{i=-\infty}^{+\infty} h[n + iN] \tag{2.6}$$

Due to the periodic nature of the impulse response, it is important to choose an MLS input with a length N that is larger than the impulse response to be measured, otherwise time-aliasing occurs. The MLS technique has been shown to present distortion artifacts or "distortion peaks" that are uniformly distributed along the deconvolved impulse response when using real-world loudspeakers and microphones [14]. In spite of this, we chose to use MLS and have our localization algorithm deal with the distortion peaks.

In 1977, Cohn and Lempel [18] found a relationship between MLS and Walsh-Hadamard matrices which led to a fast transform algorithm to obtain the impulse response from the measured MLS response.

2.3 Image-Source Model

The image-source model is a technique used to generate a synthetic room impulse response for a given sound source, sound receiver, and room. In this work, we heavily used the image-source model (ISM) to simulate sound paths taken between source and receiver in an enclosed space. Figure 2.4 shows the overall ISM method using the same room from Fig. 2.1. The main idea behind the image-source model is the generation of virtual sound sources. The path a sound reflection takes can be represented as direct sound from a virtual source. This virtual source is constructed by mirroring the location of the actual source across the rigid surface that sound reflected off of. The image-source model was first presented by Allen and Berkley in 1979 [19].

The image-source model provides a quick way to generate room impulse responses with different environmental characteristics. One drawback to Allen and Berkley's original method was that calculated time delays of impulses were rounded to the nearest time sample or bin. In 1986, Peterson [20] proposed using a sinc function at each reflection peak calculated from the Allen-Berkley's image-source model to allow exact representation of non-integer time delays similar manner to how the actual system behaves. We applied this optimization to our simulation for a more accurate room impulse response.

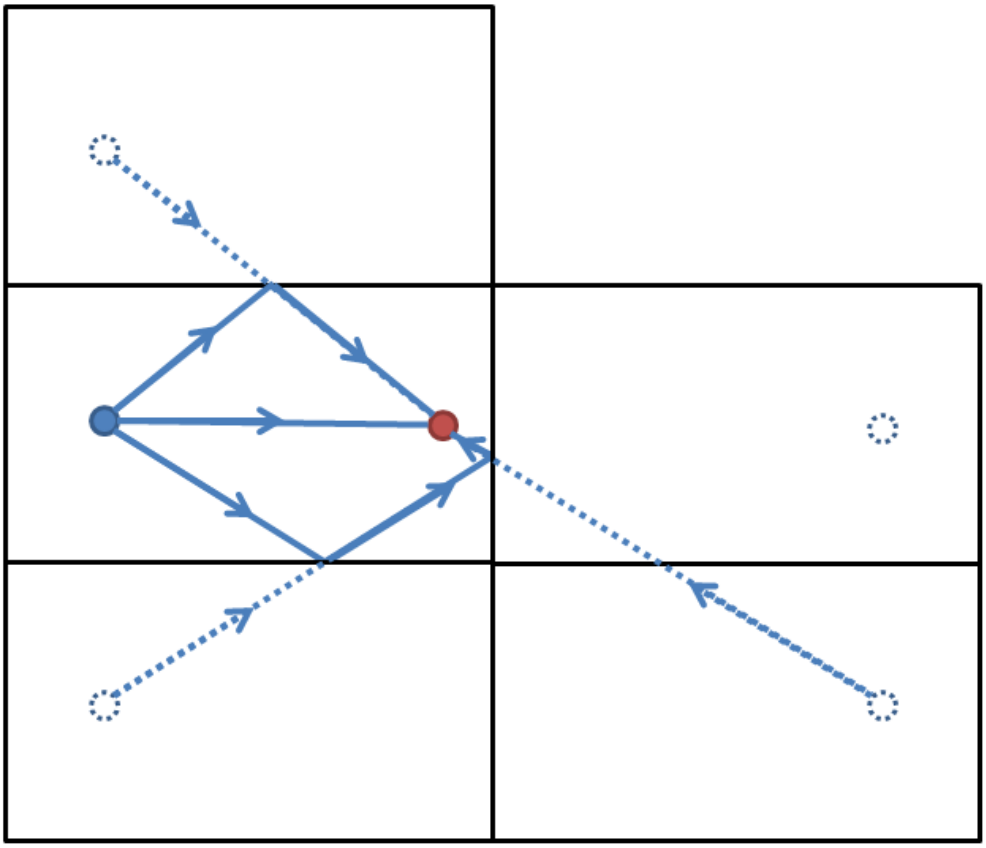

Figure 2.4: Virtual sources generated from the image-source model of the room in Fig 2.1.

2.4 Tetrahedral Microphone Array

Localization of sound sources using microphone arrays is an important and heavily researched problem. In this work, we know where the sound source and microphone array are and we want to determine the sound path the sound must have taken to get to the array from the source.

A microphone array involves using multiple microphones configured in a specific orientation operating in tandem. For our work, we assume that the microphones in the array are omnidirectional. We were interested in obtaining the locations of reflectors in 3D space so at least four microphones needed to be used. We chose a tetrahedron configuration because we wanted a compact setup and from the tetrahedral formation, we can derive the direction of arrival (DOA) of the sound in 3D space. Figure 2.5 shows an example of a tetrahedral microphone array. Let $\vec{r}_i = [x_i, y_i, z_i]$ denote the location of the

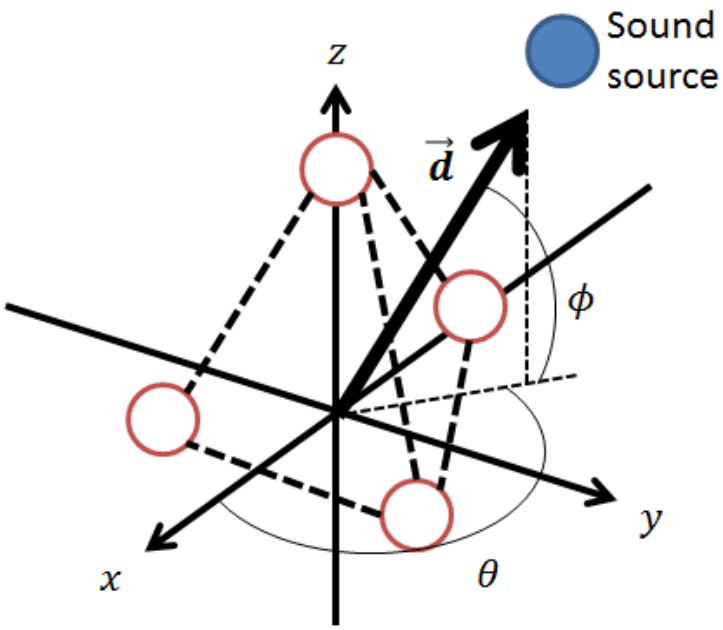

Figure 2.5: Sample tetrahedral microphone array and sound source.

ith sensor and let θ and ϕ denote the azimuth and elevation angles of the sound source, respectively. The sound source DOA vector $\vec{d}$ is given by:

$$\vec{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} \cos\phi\cos\theta \\ \cos\phi\sin\theta \\ \sin\phi \end{bmatrix} \tag{2.7}$$

CHAPTER 3

METHODS

3.1 Overall System

Figure 3.1 depicts the high-level overview of the room reconstruction process using a mobile platform and acoustics. We simulated a robot that has a loudspeaker and tetrahedral microphone array placed close together. The first step is to generate an adequate MLS signal. The MLS signal serves as the input to the room the robot will reconstruct. The resulting response is captured with the tetrahedral microphone array. For each microphone, the room impulse response is calculated through circular cross correlating the response with the input signal. These four room impulse responses are then used to localize virtual sources which are then used to compute wall/reflector locations. This information is used to update the robot's map of the room. The robot decides the next location to move to and moves to that location. This process is repeated with the new loudspeaker and microphone array locations.

3.2 Localizing Reflectors

Our method for localizing reflectors can be broken down into three steps. The first step is to determine which peaks in each RIR were caused by the same reflected sound wave and group them together. Next, we estimate the locations of the virtual sources using the grouped peak time of arrival information. Finally, we estimate the location and orientation of reflectors that produced this virtual source. In this section, we detail each component and describe the methods we chose to use.

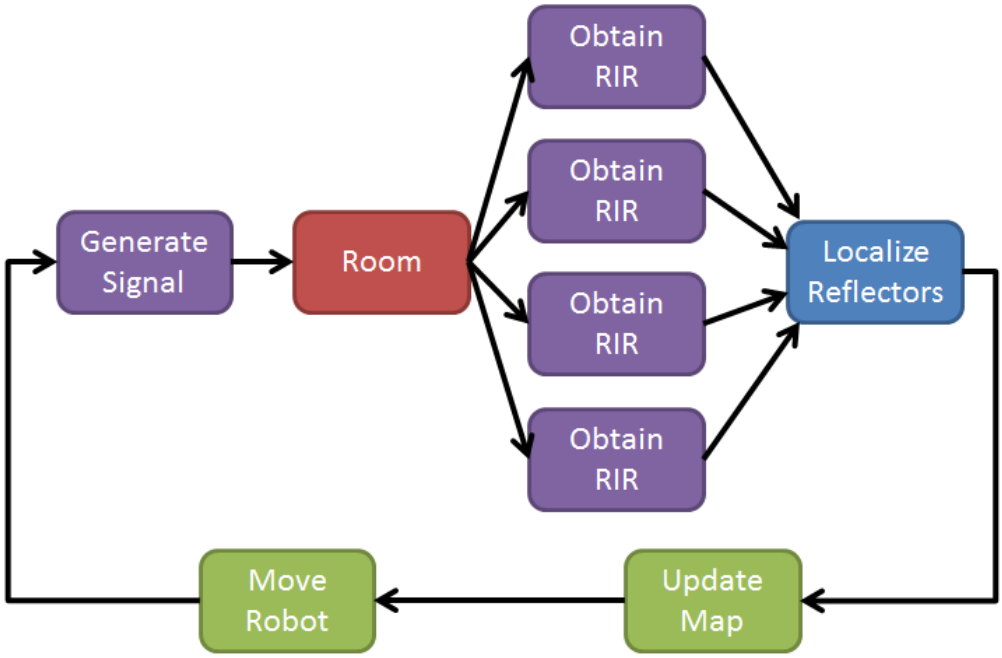

Figure 3.1: Block diagram of the overall process our robot used to navigate through the room.

3.2.1 Identifying Peak Groups

The first step to localizing virtual sources is identifying which peaks correlate with a specific virtual source. To simplify the problem, we assumed that peak groups could be identified without ambiguity. Peak group ambiguity can occur if there are two or more sound paths taken between the source and microphone array such that the sound paths are incident on the microphone array at the same time. Our assumption simplifies peak group identification to locating groups of peaks within a specific time window based on the longest path between any pair of microphones. This longest path in the tetrahedral arrangement is simply the length of an edge a. Therefore, the time window can be computed as:

$$\Delta = \frac{a}{c} \tag{3.1}$$

where c is the speed of sound in air.

Each sample index in each RIR is evaluated in lockstep until the first peak is found in any of the RIRs. Then the remaining RIRs are searched for peaks in the next $\Delta * f$ samples where f is the sampling frequency. If peaks are found in each of the remaining RIRs, then a peak group is formed consisting

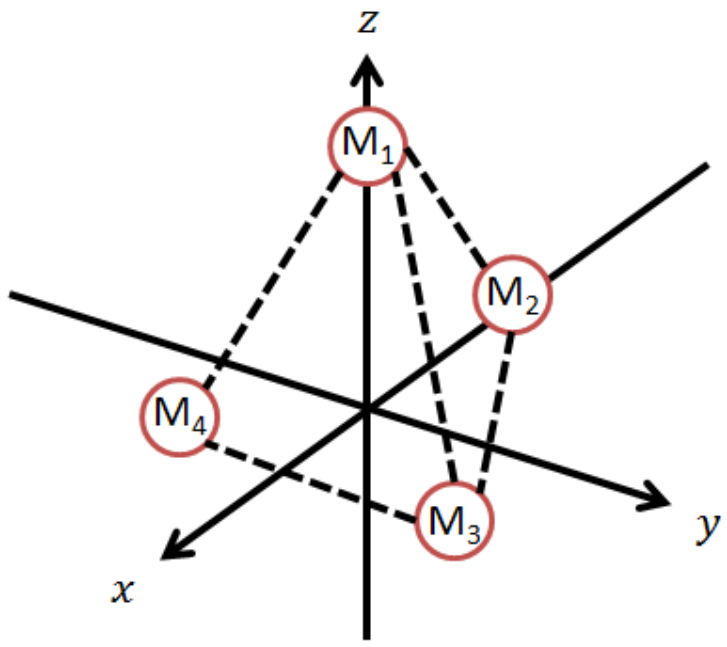

Figure 3.2: Tetrahedral microphone array with microphones labeled.

of the indexes of these four peaks. Because of our initial assumption that peak groups can be identified without ambiguity, we can remove all four peaks from the RIRs for the next peak grouping iteration. However, if there is an RIR that does not contain a peak in the time window, the initial peak is deemed spurious and removed from its respective RIR. Then the process is started again.

In addition, we chose to limit our peak group search to a distance of 6 meters or $6 * f/c$ samples.

3.2.2 Virtual Source Localization Using Spherical Wave Model

Once we have identified a peak group, we use the spherical wave model to localize the virtual source that caused these peaks. The spherical wave model treats the virtual source as an ideal point source which propagates the sound wave as a sphere [21, 22, 23]. Figure 3.2 depicts an equilateral tetrahedral microphone array. M_1, M_2, M_3, and M_4 are the microphones positioned at the vertices of the tetrahedron. Given that the center of the base of the

tetrahedron is located at $(0,0,0)$, and a is the edge length of the tetrahedron, the microphones are located at: $M_1 = (0, 0, \frac{\sqrt{6}}{3}a)$, $M_2 = (-\frac{\sqrt{3}}{3}a, 0, 0)$, $M_3 = (\frac{\sqrt{3}}{6}a, \frac{a}{2}, 0)$, and $M_4 = (\frac{\sqrt{3}}{6}a, -\frac{a}{2}, 0)$.

For a virtual source located at (x, y, z), using the spherical wave propagation model, the following equations are satisfied for each of the microphones.

$$x^2 + y^2 + (z - \frac{\sqrt{6}}{3}a)^2 = r_1^2 \tag{3.2}$$

$$(x + \frac{\sqrt{3}}{3}a)^2 + y^2 + z^2 = r_2^2 = (r_1 + r_2 - r_1)^2 \tag{3.3}$$

$$(x - \frac{\sqrt{3}}{6}a)^2 + (y - \frac{a}{2})^2 + z^2 = r_3^2 = (r_1 + r_3 - r_1)^2 \tag{3.4}$$

$$(x - \frac{\sqrt{3}}{6}a)^2 + (y + \frac{a}{2})^2 + z^2 = r_4^2 = (r_1 + r_4 - r_1)^2 \tag{3.5}$$

where r_i is the distance the sound path traveled between the sound source and the arrival at the ith microphone. r_i is determined from the RIR index n_i from a given peak: $r_i = \frac{n_i c}{f}$.

We are only concerned about x and y so solving these equations for those variables, we obtain:

$$y = \frac{-2r_1(d_{3,1} - d_{4,1}) + d_{3,1}^2 - d_{4,1}^2}{2a} \tag{3.6}$$

$$x = \frac{2r_1(d_{4,1} - d_{2,1}) + d_{4,1}^2 - d_{2,1}^2 - ay}{-\sqrt{3}a} \tag{3.7}$$

where $d_{i,j} = r_i - r_j$. When substituting in y, we obtain:

$$x = \frac{2r_1(d_{4,1} + d_{3,1} - 2d_{2,1}) + d_{4,1}^2 + d_{3,1}^2 - 2d_{2,1}^2}{-2\sqrt{3}a} \tag{3.8}$$

3.2.3 Reflector Localization from Virtual Source

For our robot, the sound source and microphone positions are known. In addition, these two are placed very close to each other because they are on the same robotic platform. We use this positioning to simplify the calculation of reflector locations. This simplification makes two assumptions.

- The distance between the original sound source and microphone array is

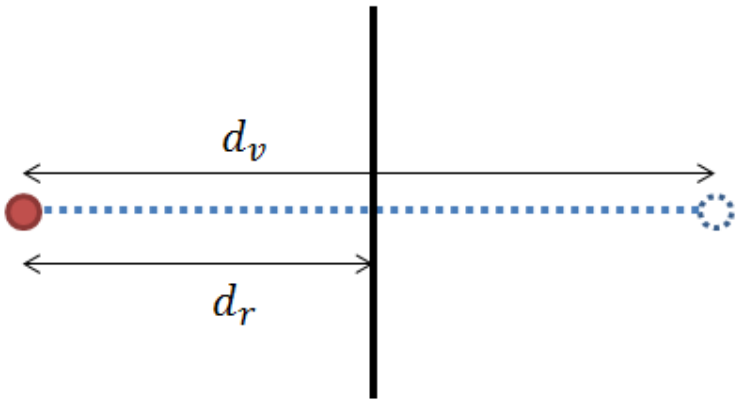

Figure 3.3: The distance d_v between the sound source (red) and virtual source (blue) created by the reflector at a distance of d_r.

very small in relation to the distance between the virtual source and the original sound source/microphone array. This allows us to assume the original sound source and microphone array are in the same location.

- The virtual source is created from only a single reflection. While this assumption may not hold true for all virtual sources, our algorithm can pick out if a second reflection occurs because the reflector location will be outside the valid area.

Given these two assumptions, if d_v is the distance between the virtual source and the real source/microphone array, the distance between the real source/microphone array and the reflector, d_r, is $d_r = \frac{d_v}{2}$. The reflector is perpendicular to the line between the virtual source and source/microphone array as shown in Fig. 3.3.

3.3 Navigation Algorithm

We chose to use the right-wall follow algorithm to map the room. We limited our simulation to closed rectangular rooms so a wall following algorithm like a right-wall follow algorithm will guarantee the walls of the room get fully explored.

3.3.1 Right-Wall Follow Algorithm

At a high-level, the right-wall follow algorithm is:

if *Opening on right* **then**
| Turn robot right
else if *Opening on front* **then**
| Do nothing
else if *Opening on left* **then**
| Turn robot left
else
| Turn robot around
end
Move Robot Forward

More specifically, once a wall to the right has been detected, the right-wall follow algorithm aims to keep the right wall at a specific distance to the right of the robot as the robot travels along the wall. In order to maintain this specific distance, a basic proportional feedback control system was used. This control system is explained in Section 3.3.2.

Since we limit our simulation to closed rectangular rooms, our right-wall follow algorithm can be simplified even further. First, if the robot ever comes to a wall in front, the robot will make a left turn and continue moving. If the robot loses the right wall, because we know the wall will always be there, we can keep the robot moving in its current direction instead of turn right and try to find the right wall.

3.3.2 Wall Distance Control System

A typical system is characterized as open-loop if the system comprises of a system that has an input, $x(t)$ and output $y(t)$. This system is typically known as a plant. A basic example of an open-loop system is shown in Fig. 3.4.

The goal of control theory is to design a system such that the plant is controlled to a desired value or trajectory. In order to control the system, we need to create a closed-loop system by using the output $y(t)$ to feedback into the input of the plant, thereby allowing the output to influence the next input. Figure 3.5 shows what a basic closed-loop system looks like.

Figure 3.4: Basic example of an open-loop system.

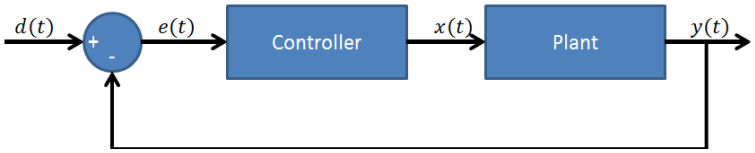

Figure 3.5: Basic example of a closed-loop system.

$d(t)$ represents the desired value or trajectory we want our output $y(t)$ to become. $e(t)$ is the error between the desired output and the actual output, or $d(t) - y(t)$. This error is the input to a custom controller we design such that the output of the controller serves as the input to the plant in order to move $y(t)$ to $d(t)$.

In our case, the plant is the distance to the right wall. We use a basic proportional controller to stabilize our system:

$$x(t) = K_p e(t) \tag{3.9}$$

where K_p is the proportional gain of the controller. K_p is essentially a tunable parameter that determines how sensitive our next input is to the current error. In our simulations, we set $K_p = 0.1$ and $d(t) = 1.5$ m.

3.4 Quadratic Interpolation

Measurement techniques to obtain the RIR typically represent the measured RIR as a discrete-time signal sampled from the continuous-time RIR. The baseline peak-picking method involves selecting the sample with the maximum value and associating that point with the peak of the signal. This method is quick and accurate if the sampling rate is high enough such that a potential half-sample discrepancy between true and measured peak times

is allowed. Another method uses discrete data interpolation to understand the continuous behavior of the underlying signal. Lai and Torp [24] used quadratic interpolation on the cross-correlation between multiple impulse responses to obtain more accurate peak times. We apply this idea to interpolate early reflection peaks in a single room impulse response. We introduce two methods for performing the quadratic peak interpolation.

Given a measured impulse response, the first step is to separate relevant sample points from the rest of the signal. We used a static threshold of 0.05 to extract these relevant sample points from the impulse response.

Three or more consecutive relevant sample points define a time-window where a reflection peak must exist. We fit the quadratic function:

$$y = c_0 x^2 + c_1 x + c_2$$

to the points in this window. For m consecutive points, this quadratic fit can be represented in matrix form as

$$\mathbf{y} = \mathbf{A}\mathbf{c}$$

where $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_m^2 & x_m & 1 \end{bmatrix}$, $\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$

We seek to find the coefficient vector $\hat{\mathbf{c}}$ of the interpolated quadratic function that minimizes the least-square error

$$\|\mathbf{y} - \mathbf{A}\mathbf{c}\|^2$$

which, given three or more points, can be calculated explicitly as:

$$\hat{\mathbf{c}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$$

The interpolated peak can be derived from $\hat{\mathbf{c}}$ using

$$\tau_i = \frac{-c_1}{2c_0}$$

for $i = 1, ..., i_{max}$, where i_{max} is the total number of peaks in the impulse response.

The next time-window is obtained and this process repeats until all time-windows have been exhausted. Because we are interested in early reflections, which are distinguishable reflections, we can assume that time-windows do not overlap.

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 Experimental Setup

The experiments were simulated using MATLAB. We used Stephen McGovern's image-source model implementation [25] as the basis for the room simulations. McGovern's model was based on Allen and Berkley's image-source model.

The overall robot simulator was designed to be modular. Each module of the simulator can be directly mapped to blocks in the overall system block diagram shown in Fig. 3.1. This way, each of the major components in the overall system — input signal generation, room simulation, RIR reconstruction, reflector localization, mapping, and robot navigation/movement — can easily be tweaked and tested without having to rewrite or change other modules.

4.2 Room Mapping Simulation

4.2.1 Setup

We simulated a robot inside a closed 10 m by 10 m by 10 m rectangular room. The robot was represented as a point, which serves as the origin of the robot's local coordinate frame. The robot also has a heading to indicate the direction the robot is facing. As mentioned before, the robot has a sound source and tetrahedral microphone array. The sound source is located 0.15 m in the direction of the robot's heading and indicated by the red circle in Fig. 4.1. The tetrahedral microphone array's edge length a was chosen to be 0.15 m and centered on the robot's origin. Each of the microphones in the

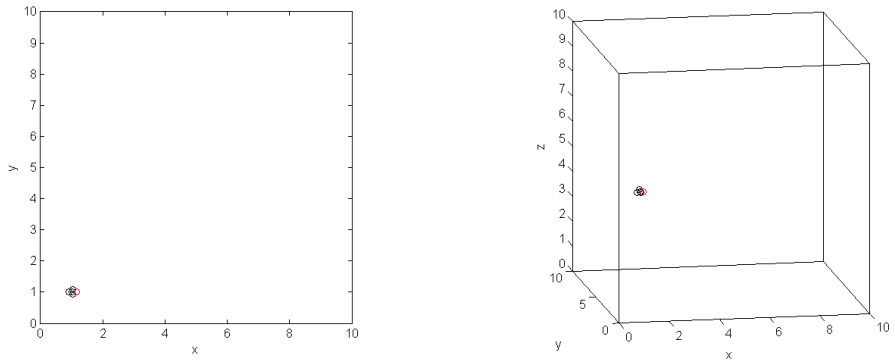

Figure 4.1: Initial simulation configuration. Room is 10 m by 10 m by 10 m and the robot is placed at (1.5 m, 1.5 m, 5 m). The blue circles represent the tetrahedral microphone locations and the red circle represents the sound source location.

microphone array are represented by blue circles in Fig. 4.1. The robot was placed initially at (1.5 m, 1.5 m, 5 m) with a heading along the x-axis.

We performed two types of simulations which differ by the type of input sound the robot uses. At the start of each timestep or iteration, the true RIRs are computed for each source and microphone pair. For the ideal case simulation, the robot receives the true RIRs as the measured response to the room, since this is what would be returned had an ideal impulse function been used as the input signal. For the MLS case simulation, the true RIRs are convolved with the selected MLS input signal to compute the measured response given to the robot. The robot computes the wall information and moves to the new location and orientation according to the navigation algorithm. Once the move is completed, the current timestep ends and a new timestep begins.

4.3 Results

We simulated the 10 m by 10 m by 10 m room and had our robot simultaneously generate a map of the room and use the map to navigate around the outer wall of the room. The robot started at location (1.5 m, 1.5 m, 5 m). The z-axis was set to 5 m to eliminate the effects of echoes from the ground and ceiling. The robot was allowed to move at most 0.3 m between timesteps. The direction was set by the navigation and control system used

to maintain the right-wall follow algorithm. We set the robot to turn if it detected a wall 1.5 m in front of the robot.

For the first set of simulations, we wanted to see how well our overall system performs in the ideal case. We assumed the input signal to the room is an ideal impulse signal so the recorded response sent to the robot was simply the true RIR computed using our modified image-source model. Figures 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, and 4.8 show the map reconstructed by the robot for 1, 10, 50, 150, 300, 500, and 1000 iterations when using the ideal RIRs to extract reflector information.

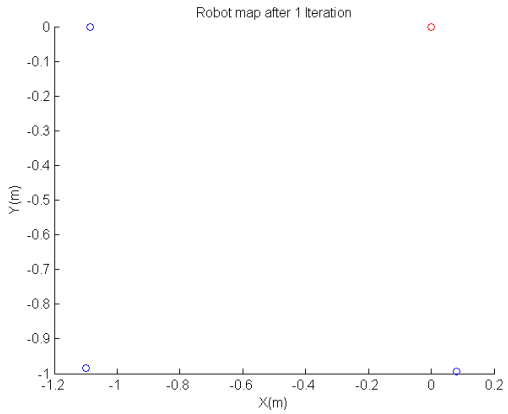

Figure 4.2: Robot's map after 1 timestep using ideal RIR.

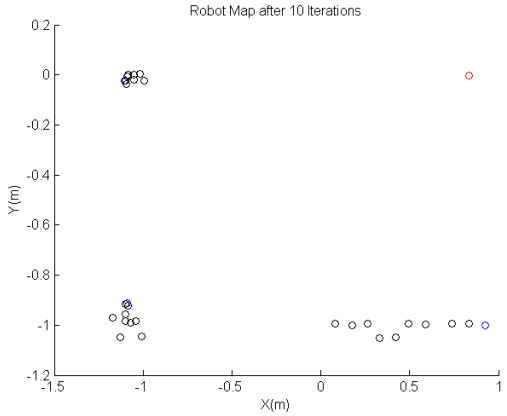

Figure 4.3: Robot's map after 10 timesteps using ideal RIR.

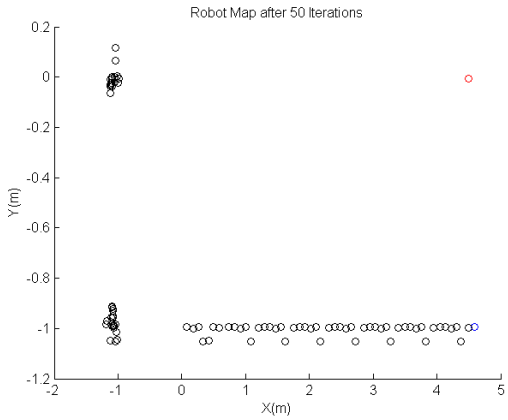

Figure 4.4: Robot's map after 50 timesteps using ideal RIR.

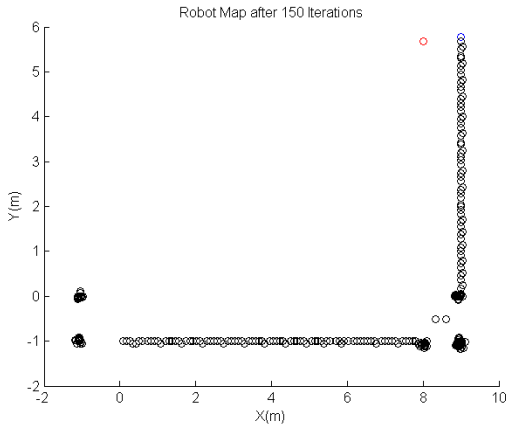

Figure 4.5: Robot's map after 150 timesteps using ideal RIR.

As we can see, the robot starts off with no information about the room and surroundings. At each timestep, information on the locations of the closest

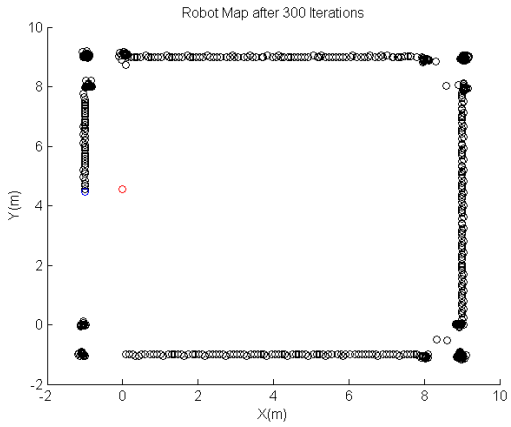

Figure 4.6: Robot's map after 300 timesteps using ideal RIR.

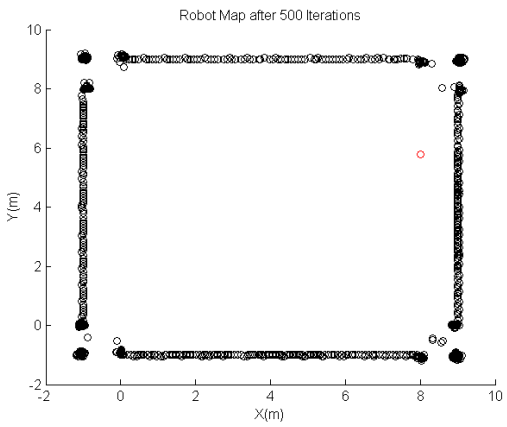

Figure 4.7: Robot's map after 500 timesteps using ideal RIR.

reflective surfaces, or in this case walls, are recorded. This information is processed to determine the next location the robot should move to.

As more timesteps occur, the robot map keeps getting updated with more information about the position of the walls. When the robot gets close to a wall in front, the robot turns 90 degrees and resumes the default right-wall following algorithm. One very important, and easily overlooked, result is the fact the overall system we designed without the acoustic reconstruction component was able to map and navigate through the room. The average root mean squared (RMS) error for each trial of 1, 10, 50, 150, 300, 500, 1000 timesteps are shown in Table 4.1. In addition, the number of data points corresponding to detected reflectors is shown.

We can see that the average RMS error is below 0.05 m which is quite accurate. In addition, once the robot has completed a circuit of the room and begins remapping known areas (500 and 1000 timesteps), the overall RMS error decreases.

Now that we obtained the ideal performance of our room reconstruction system using the ideal RIRs, we then simulated using RIRs reconstructed from sending an MLS signal through the room, recording the room response, and reconstructing the RIR. Figures 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15 show the maps reconstructed by the robot when using the MLS RIR reconstruction algorithm for the same number of timesteps as the ideal case.

The first point to note is that the robot was successful in navigating

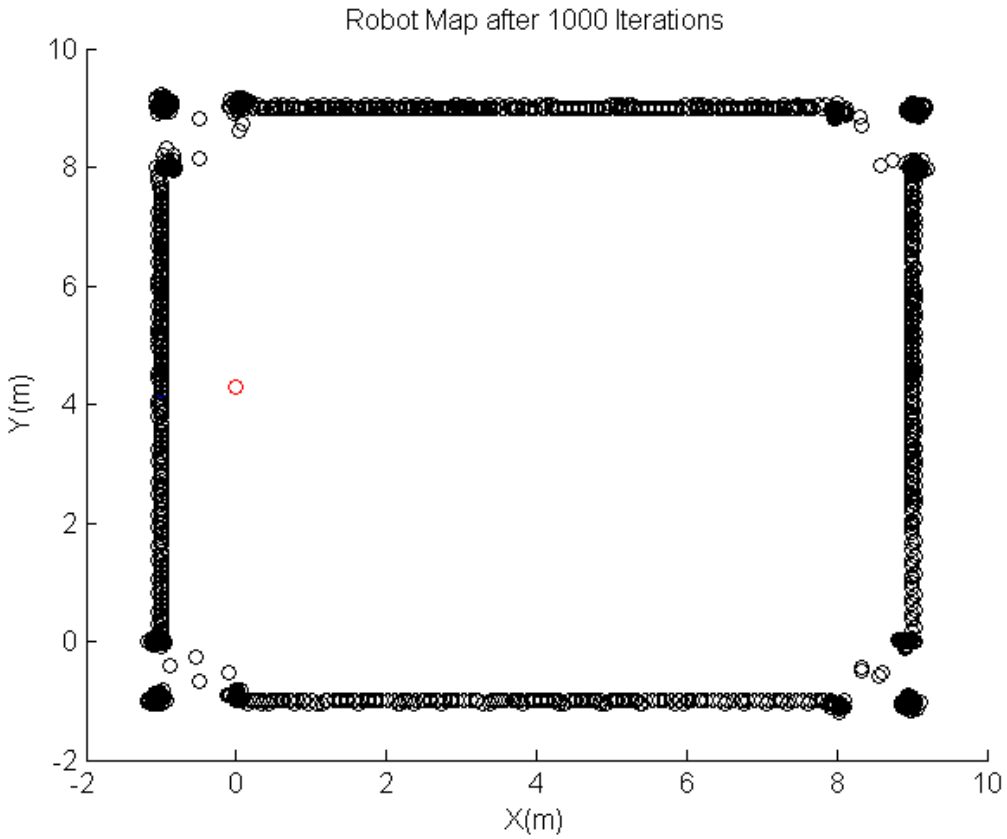

Figure 4.8: Robot's map after 1000 timesteps using ideal RIR.

Table 4.1: Average root mean square (RMS) error of the measured reflector location and the true location using ideal RIR.

Timesteps	Average RMS Error (m)	Number of Reflector Samples
1	0.0294	3
10	0.0101	30
50	0.0053	92
150	0.0455	277
300	0.0498	595
500	0.0361	966
1000	0.0275	1973

through the room using our MLS-based room reconstruction system. Looking at performance, the average root mean squared (RMS) error and number of data points corresponding to detected reflectors were recorded in Table 4.2 for each simulation of 1, 10, 50, 150, 300, 500, and 1000 timesteps. When comparing the average RMS error for the MLS reconstruction case against the ideal case, we noted that the average RMS error for the MLS case was

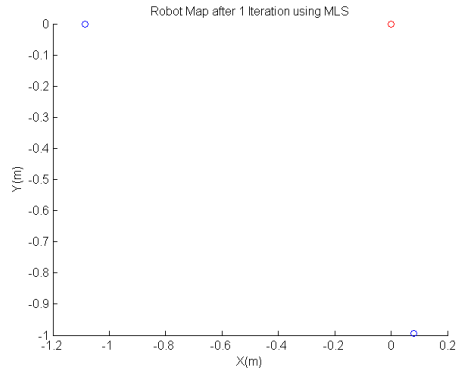

Figure 4.9: Robot's map after 1 timestep using RIR reconstructed from MLS algorithm.

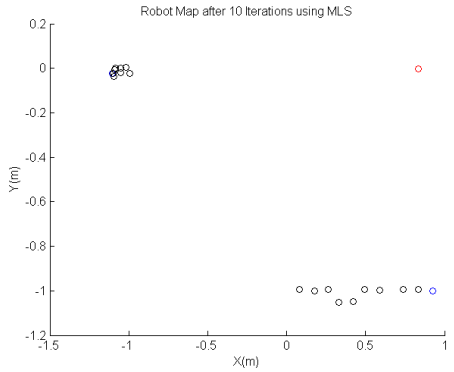

Figure 4.10: Robot's map after 10 timesteps using RIR reconstructed from MLS algorithm.

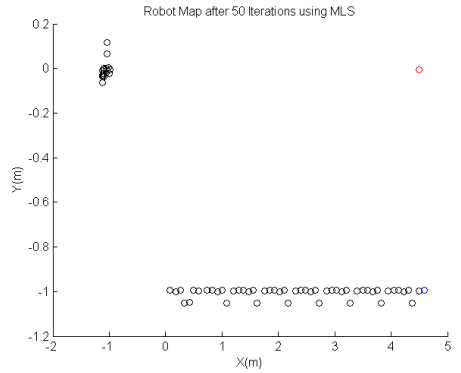

Figure 4.11: Robot's map after 50 timesteps using RIR reconstructed from MLS algorithm.

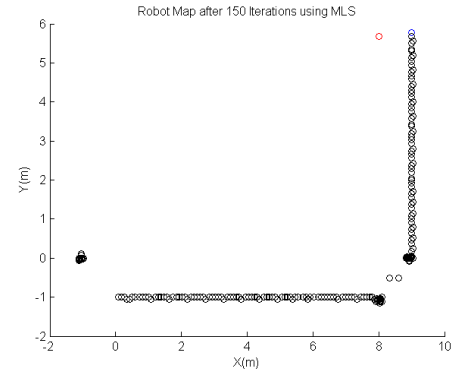

Figure 4.12: Robot's map after 150 timesteps using RIR reconstructed from MLS algorithm.

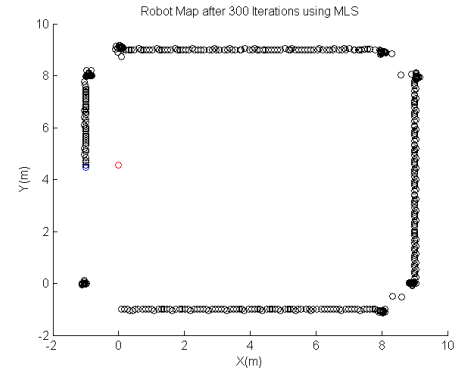

Figure 4.13: Robot's map after 300 timesteps using RIR reconstructed from MLS algorithm.

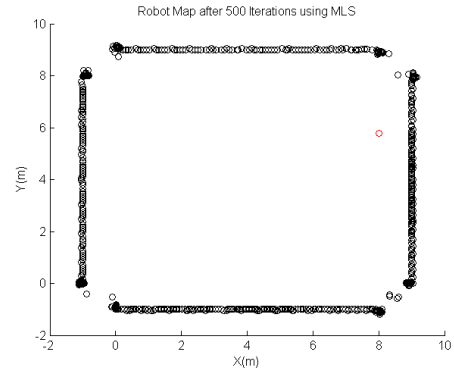

Figure 4.14: Robot's map after 500 timesteps using RIR reconstructed from MLS algorithm.

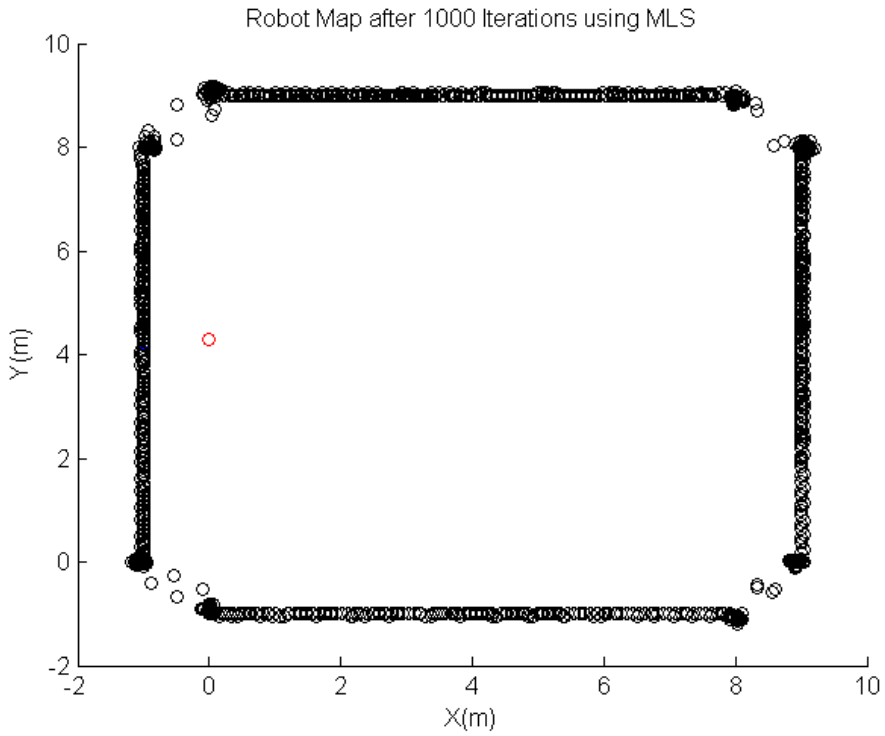

Figure 4.15: Robot's map after 1000 timesteps using RIR reconstructed from MLS algorithm.

Table 4.2: Average root mean square (RMS) error of the measured reflector location and the true location using RIRs reconstructed with the MLS algorithm.

Timesteps	Average RMS Error (m)	Number of Reflector Samples
1	0.0434	2
10	0.0127	20
50	0.0059	72
150	0.0543	216
300	0.0555	454
500	0.0410	743
1000	0.0311	1507

still below 0.06 m. It should be noted, however, that the different in the number of reflector samples obtained from the MLS reconstruction versus the ideal RIR case increased as the timesteps increased, however the end result mapping looks very similar.

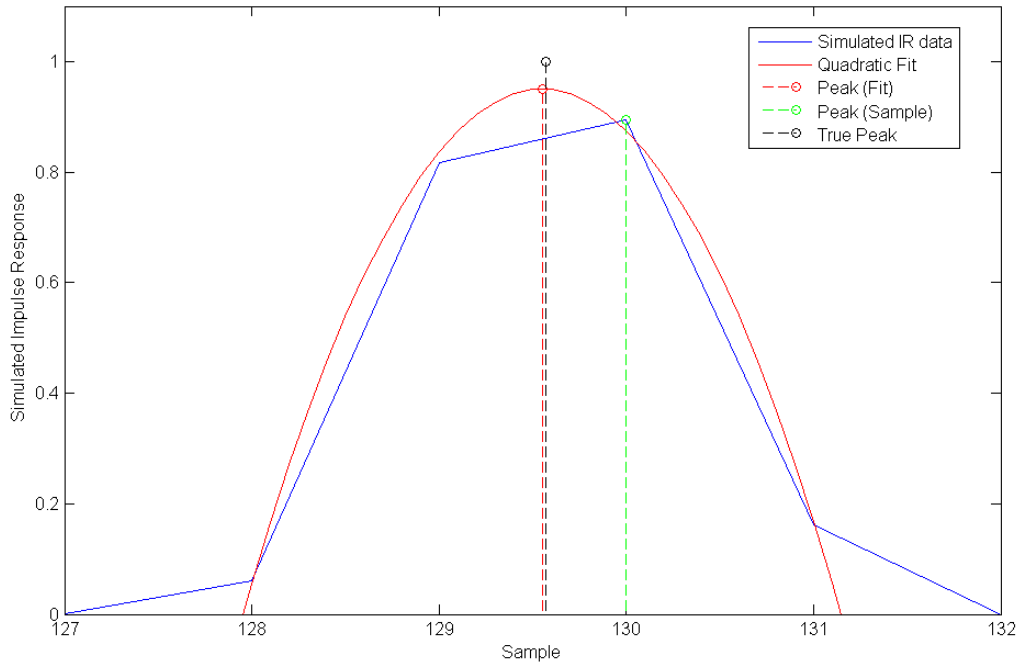

Figure 4.16: Graphical evaluation of the quadratic interpolation method vs. the baseline peak-picking method on a simulated ideal room impulse response peak.

4.4 Quadratic Interpolation Simulation

4.4.1 Setup

We simulated a 4 m by 4 m by 4 m room with the receiver placed at (2 m, 2 m, 2 m) and sound source at (3 m, 2 m, 2 m) using 9 and 127 virtual sources. Simulations were carried out using Stephen McGovern's image-source model implementation which was based on Allen and Berkley's image-source model [25]. We augmented McGovern's simulation using Peterson's sinc function method [20] by using:

$$s[n] = sinc(\tfrac{3}{5}\pi(nT - \tau_i))$$

for $\tau - 2 < n < \tau + 2$.

4.4.2 Results

We simulated a 4 m by 4 m by 4 m room with the receiver placed at (2 m, 2 m, 2 m) and sound source at (3 m, 2 m, 2 m) using 9 and 729 virtual

Table 4.3: Average root mean square (RMS) error between the real and calculated arrival time of RIR peaks gathered from an ideal RIR in a simulated room.

Method	Average RMS Error (sample)	# of Virtual Sources
Baseline	0.3002	9
Quadratic	0.0281	9
Baseline	0.2990	729
Quadratic	0.0276	729

Table 4.4: Average root mean square (RMS) error between the real and calculated arrival time of RIR peaks gathered from an MLS-computed RIR in a simulated room.

Method	Average RMS Error (sample)	# of Virtual Sources
Baseline	0.2990	9
Quadratic	0.0344	9
Baseline	0.3199	729
Quadratic	0.0435	729

sources. The quadratic peak interpolation method was compared against the baseline peak-picking method and the results are shown in Table 4.3. Figure 4.16 demonstrates both methods used on a single simulated impulse response peak. The results show that the per-peak RMS error of our quadratic interpolation method is about a factor of ten less than the baseline method.

We also obtained an estimate of the simulated impulse response using a maximum length sequence of order 15 and applied both peak-picking methods. The results are shown in Table 4.4. Again, our quadratic interpolation method performed better than the baseline method by an order of magnitude.

We also applied our quadratic interpolation method to measured RIR data. Figure 4.17 shows the measured impulse response and Fig. 4.18 and 4.19 show the baseline method and our method applied to two different measured impulse response peak data. Evaluating the quality of the interpolated peak against the baseline method cannot be done because there is no ground-truth data.

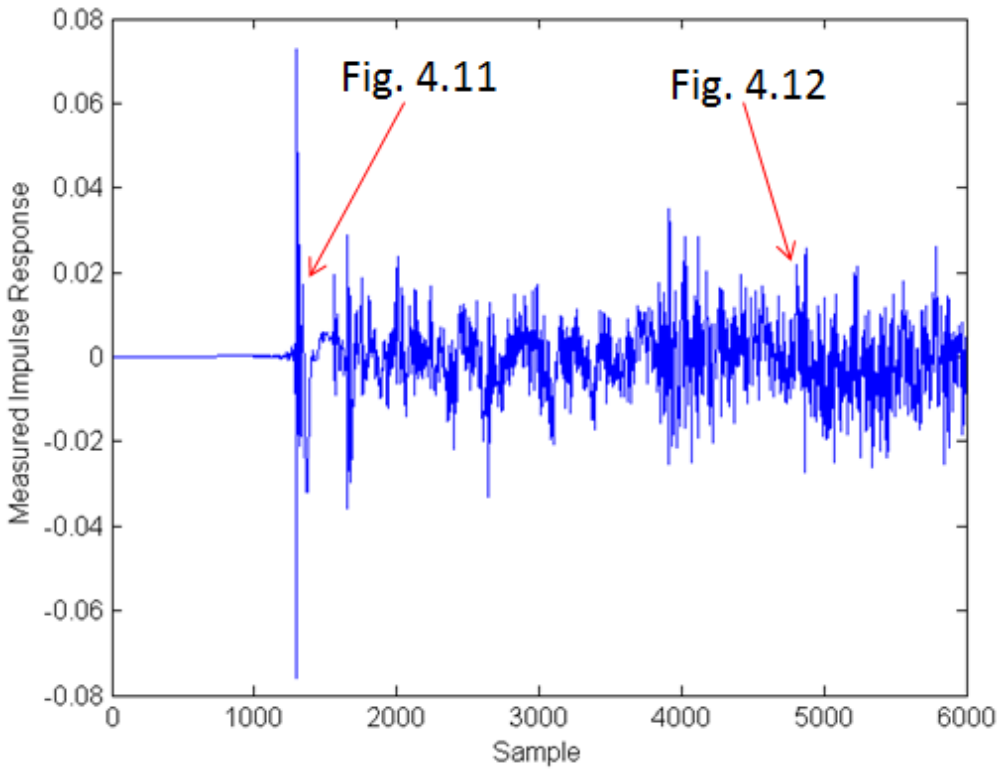

Figure 4.17: Example of a measured room impulse response. The arrows indicate the peaks explored in Fig. 4.18 and 4.19.

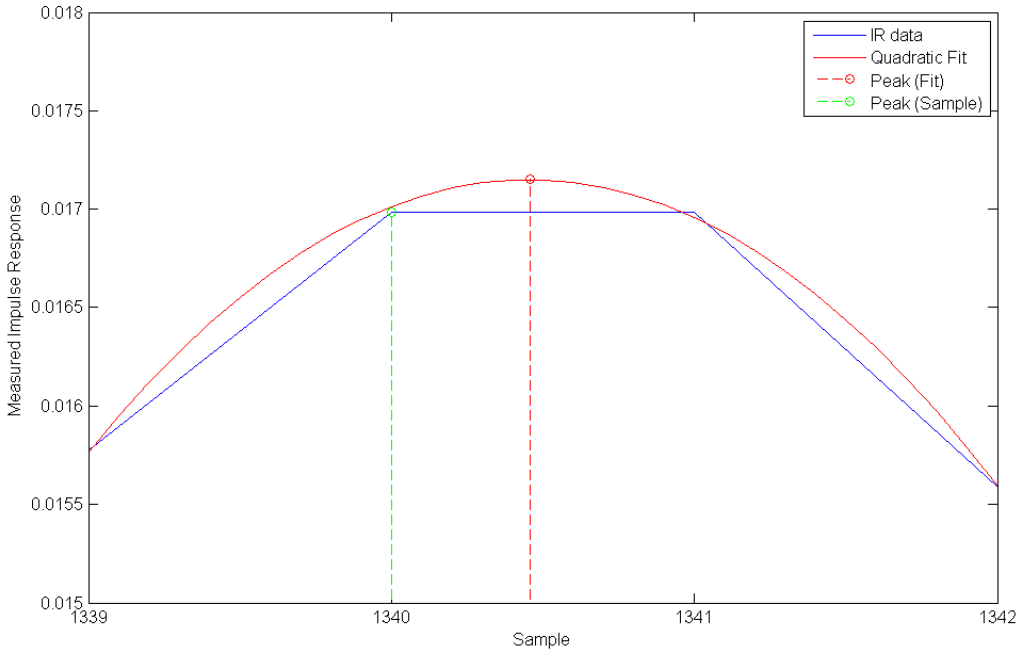

Figure 4.18: Graphical evaluation of the quadratic interpolation method vs. the baseline peak-picking method on a measured impulse response peak.

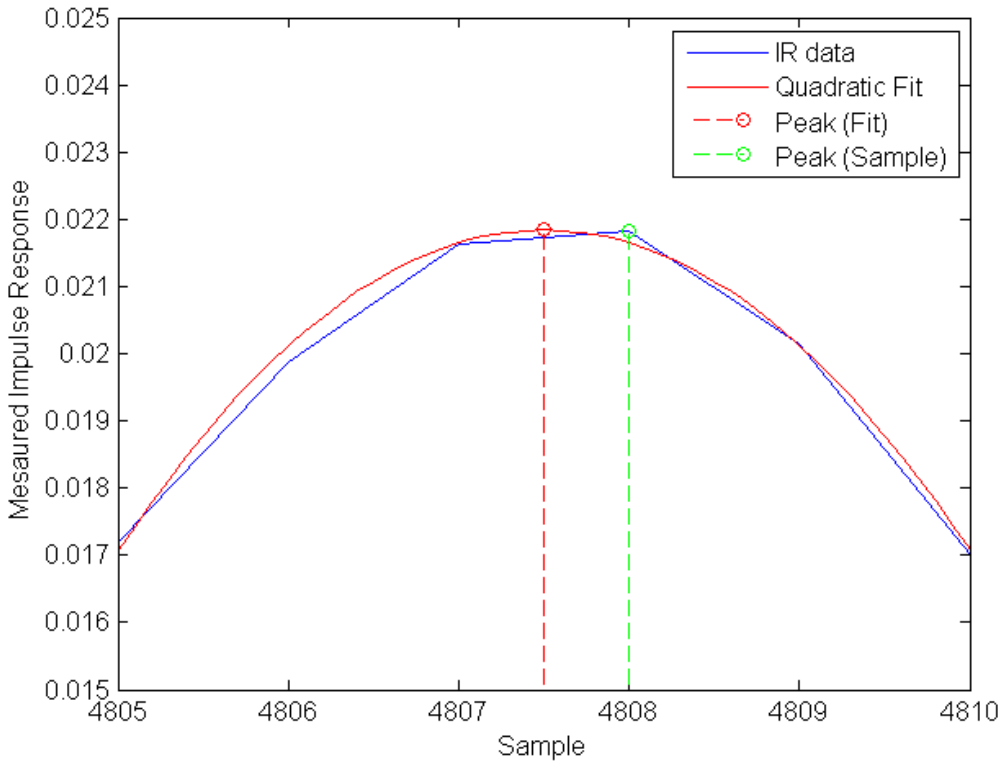

Figure 4.19: Graphical evaluation of the quadratic interpolation method vs. the baseline peak-picking method on a measured impulse response peak.

CHAPTER 5

DISCUSSION AND CONCLUSIONS

5.1 Room Mapping

Important information about the validity of room reconstruction using room impulse responses is gained from the results of these simulations. In addition, these results show that room impulse responses obtained using active sonar techniques and a unique sound input are accurate enough to be used for room reconstruction. Table 5.1 shows a summary of our results.

The first point to note is the maximum average RMS error in the estimated room reconstruction obtained using the ideal RIR was 0.0498 m. The maximum average RMS error in the estimated room reconstruction obtained using an RIR obtained from the MLS algorithm was 0.0555 m. In both cases, the average RMS error was about 5 cm, which indicates RIR-based reconstruction would be good for mapping bigger spaces.

The average difference between the error in the ideal RIR simulations and the error in the reconstructed RIR via MLS algorithm was 0.57 cm. The average difference is low which means the reflection points obtained from the MLS reconstruction RIR are accurate with respect to the reflection points obtained from the ideal RIR.

It is important to note, however, that while using MLS reconstruction, about 74% was the average percent of the ideal RIR data points were recorded. This percent was always above 66% for all the trials. These values were a result of the threshold we set when segmenting echo samples from non-echo samples in the reconstructed RIR. We chose a conservative threshold to avoid spurious echo detection from noise. With fine-tuning, the percentage of MLS reconstructed RIR echoes correlating with ideal RIR echoes should increase. Even with our conservative threshold, we were able to obtain a recognition rate of at least 66% and an average of 74%, both of which are more than

Table 5.1: Difference between average root mean square (RMS) error of the measured reflector location and the true location using ideal RIR, and ratio of the number of MLS reflector samples to ideal reflector samples.

Timesteps	Difference in Average RMS Error (m)	Ratio of MLS reflector samples to ideal reflector samples
1	0.014	0.667
10	0.0026	0.667
50	0.0006	0.783
150	0.0088	0.780
300	0.0057	0.763
500	0.0049	0.769
1000	0.0036	0.764
Average	0.0057	0.742

enough to construct a viable map for the robot to use to navigate.

5.2 Quadratic Interpolation

The results show that our quadratic interpolation method for sampled room impulse responses improves the peak-picking resolution by an order of magnitude over the baseline peak-picking method. This allows our simulation of the echo peak locations in the RIR to be even more accurate. The increased accuracy of the RIR peak locations directly improves our reflector localization algorithm, which improves our robot's map of the room.

5.3 Conclusion

In this work, we created a simulation of a robot using MLS to estimate the impulse response of the environment, which in turn was used to reconstruct the environment around the robot and ultimately used as a map for robot navigation. This simulation yielded a number of important results. First, we demonstrated a method for a robot to estimate walls in the room it is in with about 5 cm accuracy if the room impulse response is known. Second, we showed that a robot can use sound to actively obtain this room impulse response and only incur about a 0.57 cm difference from the ideal case. In addition, we showed that the mappings generated from the room

can be used to navigate the robot through the room. Finally, we showed that we can improve on the error in accuracy by an order of magnitude using quadratic interpolation to obtain subsample accuracy on the echo peaks in the RIR, which translates directly to more accurate reflector localization and ultimately more accurate data points on our map.

5.4 Future Work

Due to the modularity of the simulation design, each component can be addressed and improved on individually, which will improve the accuracy of the simulation as a whole. First, further simulations of the different types of RIR reconstruction methods could yield more accurate RIRs to work from. Another area is improving the peak detection algorithm used in each RIR. Improvements can also be done to increase the peak set recognition algorithm which will increase the accuracy of virtual sources localization. There are also a large number of source localization algorithms that could be applied to localize the virtual source more accurately. Not only that, there are a number of algorithms to increase the accuracy of reflector locations. The mapping and navigation algorithm is another big area to explore.

Another direction for future work would be to outfit a real robot with a loudspeaker and tetrahedral microphone array. This would enable us to see the validity of the simulation work in the real world.

Finally, further work could go into merging multiple sensors in with this data. The data obtained from the tetrahedral microphone array is by nature sparse and acquired fairly quickly. Combining this sparse and rough outline of the environment with other more computationally or data intensive sensors can allow the robot to save computation time and data space by allowing the robot to target only specific areas that need a more detailed view that the acoustic reconstruction could not provide.

APPENDIX A

ROOM SIMULATION CODE

A.1 Execute Script

```
%EXECUTE  Execute Simulation Script
%   This Matlab script provides the main flow
%   of the simulation. Edit general configuration
%   settings in the file to specify the conditions
%   under which the simulation should run.
%
%   mls_flag        = Flag for MLS reconstruction
%                     1 is MLS, 0 is ideal.
%   mls_normalize   = Flag to normalize MLS
%                     reconstructed RIR - 1 is MLS,
%                     0 is ideal.
%   iterations      = Number of timesteps or
%                     iterations the simulation
%                     will run for.
%   g_robot_x       = Starting x location of the
%                     robot (meters)
%   g_robot_y       = Starting y location of the
%                     robot (meters)
%   g_robot_angle   = Starting angle of the
%                     robot (degrees)
%   fs              = Sampling rate
%   n               = Determines number of virtual
%                     sound sources
%                     (2*n+1)^3 number of virtual
%                     sources generated
```

```
%   r               = Reflection coefficient
%                     (-1 < r < 1)
%   rm              = Room dimensions (meters)
%
%   Note:
%       1.) All distances are in meters and all
%           angles are in degrees.
%
%   Created by: Christopher Co
%
%   Copyright 2012 Christopher Co


% General Configuration Settings

mls_flag = 1;       % Enable MLS RIR reconstruction
mls_normalize = 1;  % Normalize MLS RIR

iterations = 10;    % Number of timesteps in simulation

g_robot_x = 1;      % Current robot x-position in
                    %   world POV
g_robot_y = 1;      % Current robot y-position in
                    %   world POV
g_robot_angle = 0;  % Current robot angle in world POV

fs = 44100;         % Sampling frequency
n = 2;              % (2*n+1)^3 virtual sources
r = 0.2;            % Reflection coefficient
rm = [10 10 10];    % Room dimensions


% Variables

total_error = 0;    % Total cumulative RMS error
total_count = 0;    % Total number of data points
```

```
robot_x = 0;        % Current robot x-position in
                    %   robot's POV
robot_y = 0;        % Current robot y-position in
                    %   robot's POV
robot_angle = 0;    % Current robot angle in robot's
                    %   POV

world_x = -1000;    % X coordinates of all data
                    %   points in world POV
world_y = -1000;    % Y coordinates of all data
                    %   points in world POV

turn_flag = 0;      % Flag to track the number
                    %   of robot turns


% Set Up Figures

fig3 = figure(3);
winsize3 = get(fig3,'Position');
winsize3(1:2) = [0 0];
numframes = 9;
A3=moviein(numframes,fig3,winsize3);
set(fig3,'NextPlot','replacechildren');

fig1 = figure(1);
winsize = get(fig1,'Position');
winsize(1:2) = [0 0];
numframes = 9;
A=moviein(numframes,fig1,winsize);
set(fig1,'NextPlot','replacechildren');


% Start Simulation Loop
```

```
for i = 1:iterations
    % Compute the location of the sound source
    % based on the actual robot angle
    mic=[g_robot_x g_robot_y 5];
    src = mic;
    src_d = 0.15;
    src(1) = src(1)+src_d*cos(g_robot_angle*pi/180);
    src(2) = src(2)+src_d*sin(g_robot_angle*pi/180);
    clf(fig1);

    % Generate RIRs
    [RIR1, RIR2, RIR3, RIR4]= generate_room(fs,
            mic, n, r, rm, src, g_robot_angle,
            mls_flag, mls_normalize);
    A(:,i) = getframe(fig1,winsize);

    % Localize Reflectors
    [wall_x, wall_y] = localize_reflectors(RIR1,
            RIR2, RIR3, RIR4, mls_flag);

    % Calculate RMS error of data points
    [error, count] = compute_error(wall_x,
            wall_y, g_robot_x, g_robot_y);
    total_error = total_error + error;
    total_count = total_count + count;

    % Update the robot's current map
    clf(fig3);
    [world_x, world_y] = robot_map(wall_x, wall_y,
            robot_x, robot_y, robot_angle,
            world_x, world_y);
    A3(:,i) = getframe(fig3,winsize3);

    % Move robot
    [robot_x, robot_y, robot_angle,
    g_robot_x, g_robot_y, g_robot_angle,
```

```
    turn_flag] = robot_update(world_x, world_y,
                              robot_x, robot_y,
                              robot_angle, wall_x,
                              wall_y, g_robot_x,
                              g_robot_y, g_robot_angle,
                              turn_flag);
end

% Compute Average RMS Error
avg_error = sqrt(total_error)/total_count;
```

A.2 Room/RIR Creation Functions

```
function [out1, out2,
out3, out4]=generate_room(fs, mic, n, r, rm,
                          src, angle, mls,
                          mls_normalize);
%GENERATE_ROOM  Generates RIRs for the
%               tetrahedral microphone array
%   [out1, out2,
%       out3, out4] = GENERATE_ROOM(fs, mic,
%                          n, r, rm, src,
%                          angle, mls,
%                          mls_normalize)
%           creates four RIRs based on the
%           location of the sound source and
%           each of the microphones in the
%           tetrahedral array.  MLS
%           reconstruction of these RIRs
%           are done if the mls_flag is set.
%
%   fs              = Sampling rate
%   mic             = Origin of tetrahedral
%                       microphone array
%   n               = Number of virtual sources
%                       calculated (2*n+1)^3
%   r               = Reflection coefficient
%                       of the walls (-1 < r < 1)
%   rm              = Room dimensions (meters)
%   src             = Location of sound source
%   angle           = Robot angle in world
%                       POV (degrees)
%   mls             = Flag to enable MLS
%                       RIR reconstruction
%   mls_normalize   = Flag to enable MLS
%                       RIR normalization
%
```

```
%   Created by: Christopher Co
%
%   Copyright 2012 Christopher Co


% Variables

edge = 0.15;    % Length of edge of
                %   tetrahedral microphone
                %   array
mic_d = sqrt(3)/3*edge;


% Show source location in room figure
figure(1);
plot3(src(1), src(2), src(3), 'ro');
hold on;


% Create MLS input signal if MLS flag is set
if mls
    mls_num = 15;
    mls_input = mls_new(mls_num);
end


% Mic 1

% Compute Mic 1 location
temp = mic;
temp(3) = temp(3)+sqrt(6)/3*edge;
plot3(temp(1), temp(2), temp(3), 'ko');

% Compute ideal RIR
out1 = rir(fs, temp, n, r, rm, src);
```

```
% Compute MLS reconstruction of RIR if
% MLS flag is set
if mls
    out1 = do_mls_rir(mls_input,
            out1, mls_normalize);
end


% Mic 2

% Compute Mic 2 location
temp = mic;
temp(1) = temp(1)+mic_d*cos((180+
                    angle)*pi/180);
temp(2) = temp(2)+mic_d*sin((180+
                    angle)*pi/180);
plot3(temp(1), temp(2), temp(3), 'ko');

% Compute ideal RIR
out2 = rir(fs, temp, n, r, rm, src);

% Compute MLS reconstruction of RIR
% if MLS flag is set
if mls
    out2 = do_mls_rir(mls_input,
            out2, mls_normalize);
end


% Mic 3

% Compute Mic 3 location
temp = mic;
temp(1) = temp(1)+mic_d*cos((60+
                    angle)*pi/180);
temp(2) = temp(2)+mic_d*sin((60+
```

```
                    angle)*pi/180);
plot3(temp(1), temp(2), temp(3), 'ko');

% Compute ideal RIR
out3 = rir(fs, temp, n, r, rm, src);

% Compute MLS reconstruction of RIR
% if MLS flag is set
if mls
    out3 = do_mls_rir(mls_input,
            out3, mls_normalize);
end


% Mic 4

% Compute Mic 4 location
temp = mic;
temp(1) = temp(1)+mic_d*cos((-60+
                angle)*pi/180);
temp(2) = temp(2)+mic_d*sin((-60+
                angle)*pi/180);
plot3(temp(1), temp(2), temp(3), 'ko');

% Compute ideal RIR
out4 = rir(fs, temp, n, r, rm, src);

% Compute MLS reconstruction of RIR
% if MLS flag is set
if mls
    out4 = do_mls_rir(mls_input,
            out4, mls_normalize);
end


% Add axes and labels to room figure
```

```
axis equal;
axis([0 rm(1) 0 rm(2) 0 rm(3)]);
box on;
xlabel('x');
ylabel('y');
zlabel('z');
end
```

```
function [out, rounded]=rir(fs, mic, n, r, rm, src);
%RIR   Room Impulse Response.
%   [h] = RIR(FS, MIC, N, R, RM, SRC) performs a room
%         impulse response calculation by means of
%         the mirror image method.
%
%      FS  = sample rate.
%      MIC = row vector giving the x,y,z coordinates
%            of the microphone.
%      N   = The program will account for (2*N+1)^3
%            virtual sources
%      R   = reflection coefficient for the walls,
%            in general -1<R<1.
%      RM  = row vector giving the dimensions of
%            the room.
%      SRC = row vector giving the x,y,z coordinates
%            of the sound source.
%
%   EXAMPLE:
%
%      >>fs=44100;
%      >>mic=[19 18 1.6];
%      >>n=12;
%      >>r=0.3;
%      >>rm=[20 19 21];
%      >>src=[5 2 1];
%      >>h=rir(fs, mic, n, r, rm, src);
%
%   NOTES:
%
%   1) All distances are in meters.
%   2) The output is scaled such that the largest
%      value of the absolute value of the output
%      vector is equal to one.
%   3) To implement this filter, you will need to
%      do a fast convolution.  The program FCONV.m
```

```
%      will do this. It can be found on the Mathworks
%      File Exchange at www.mathworks.com/matlabcentral
%      /fileexchange/.  It can also be found at
%      http://www.sgm-audio.com/research/rir/fconv.m
%   4) A paper has been written on this model.  It is
%      available at:
%      http://www.sgm-audio.com/research/rir/rir.html
%
%
%Version 3.4.2
%Copyright  2003 Stephen G. McGovern

%Some of the following comments are references to
%equations the my paper.

nn=-n:1:n;                              % Index for
                                        % the sequence
rms=nn+0.5-0.5*(-1).^nn;                % Part of
                                        % equations 2,3,& 4
srcs=(-1).^(nn);                        % part of
                                        % equations 2,3,& 4
xi=srcs*src(1)+rms*rm(1)-mic(1);        % Equation 2
yj=srcs*src(2)+rms*rm(2)-mic(2);        % Equation 3
zk=srcs*src(3)+rms*rm(3)-mic(3);        % Equation 4

[i,j,k]=meshgrid(xi,yj,zk);             % convert vectors
                                        % to 3D matrices
d=sqrt(i.^2+j.^2+k.^2);                 % Equation 5
time=round(fs*d/343)+1;                 % Similar to
                                        % Equation 6
index = union(time(:),time(:));
rawtime = (fs*d/343)+1;
rawtime = sort(union(rawtime(:),rawtime(:)));

[e,f,g]=meshgrid(nn, nn, nn);           % convert vectors
                                        % to 3D matrices
```

```
c=r.^(abs(e)+abs(f)+abs(g));          % Equation 9
e=c./d;                               % Equivalent to
                                      % Equation 10

h=full(sparse(time(:),1,e(:)));       % Equivalent
                                      % to equation 11
h=h/max(abs(h));                      % Scale output

% -- START CHRIS EDIT -- %
% Output list of peaks instead of the impulse response h
%listh = h(index);
out = h;
%out = [listh rawtime];
% -- END CHRIS EDIT -- %
```

A.3 Localize Reflectors Functions

```
function [wall_x, wall_y] =
    localize_reflectors(RIR1, RIR2, RIR3, RIR4, mls)
%LOCALIZE_REFLECTORS    Determines the location of
%           the walls the sound reflected off of
%           before hitting the microphone array
%
%   [wall_x, wall_y] = LOCALIZE_REFLECTORS(RIR1,
%       RIR2, RIR3, RIR4, mls) computes the wall
%       locations that the sound reflected off of.
%       The function forms peak groups from the
%       input RIRs and computes the location of
%       the virtual sources that created the time
%       delays of each peak in the peak group.
%       The location of the reflector is then
%       calculated from the location of the robot
%       and the virtual source. The computed wall
%       x and y locations are then saved in an array
%       and returned.
%
%   RIR1    = Input RIR from microphone 1
%   RIR2    = Input RIR from microphone 2
%   RIR3    = Input RIR from microphone 3
%   RIR4    = Input RIR from microphone 4
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Initialize Variables
first = 0;          % Flag to detect direct sound
num_wall = 1;       % Number of walls (used as a
                    %   counter/indexer into wall array)
```

```
tempRIR1 = RIR1;
tempRIR2 = RIR2;
tempRIR3 = RIR3;
tempRIR4 = RIR4;


% Set up figure 2 to plot computed locations of
% reflectors in robot's POV
figure(2);
clf(2);
plot(0, 0, 'ro');
hold on;


% Start loop
while 1
    % Find peak group
[idx1, idx2, idx3, idx4, tempRIR1, tempRIR2,
    tempRIR3, tempRIR4, done] = find_peak_group(tempRIR1,
                    tempRIR2, tempRIR3, tempRIR4, mls);

    % Check if we're done
    if done == 1
break;
    end

    % Skip first peak group - this is direct sound
    if first == 0
        first = 1;
        continue
    end

    % Compute virtual source location
[v_x, v_y] = compute_virtual_source(idx1,
                        idx2, idx3, idx4);
```

```
    % Compute reflector location
[w_x, w_y] = compute_wall(v_x, v_y);

    % Update plot and wall coordinates array
    plot(w_x, w_y, 'o');
    wall_x(num_wall) = w_x;
    wall_y(num_wall) = w_y;
    num_wall = num_wall+1;
end

end
```

```
function [idx1, idx2, idx3, idx4, outRIR1, outRIR2,
outRIR3, outRIR4, done] = find_peak_group(RIR1,
RIR2, RIR3, RIR4, mls);
%FIND_PEAK_GROUP Parses the RIRs and groups
%                peaks together
% [idx1, idx2, idx3, idx4, outRIR1,
%   outRIR2, outRIR3, outRIR4,
%   done] = FIND_PEAK_GROUP(RIR1, RIR2, RIR3,
%           RIR4, mls) parses the RIRs and
%       calculates the peak groups using the time
%       delays of the peaks in the RIR.  The
%       function iterates through all four RIRs in
%       lockstep.  When a peak is identified in any
%       of the RIRs, a time-window is started.
%       The other RIRs are searched for peaks in
%       that time-window.  If all RIRs have a peak in
%       the time-window, the peaks are grouped
%       together and recorded.  Otherwise, the grouping
%       is thrown out and the first iterator starts
%       again to find the beginning of a peak group.
%
% RIR1 = Input RIR from microphone 1
% RIR2 = Input RIR from microphone 2
% RIR3 = Input RIR from microphone 3
% RIR4 = Input RIR from microphone 4
% mls = Flag signifying whether MLS reconstructed
%               RIRs were used
%
% Created by: Christopher Co
%
%   Copyright 2012 Christopher Co


% Initialize Variables
done = 0; % Flag to indicate we
                        %   are finished
```

```
idx = 1; % Starting index
finish_thresh = 771;    % Threshold to signify
                        %   the end of the search
                        %   for peak
                        % groups (in samples)
array_thresh = 25;      % Time-window (in samples)

thresh = 0; % Threshold used to de-noise
                        %   ideal RIR
if mls                  % Set threshold to de-noise
                        %   MLS reconstructed RIR
thresh = 0.002;
end

outRIR1 = RIR1;
outRIR2 = RIR2;
outRIR3 = RIR3;
outRIR4 = RIR4;

% Start loop
while 1
    % End condition
    if idx >= finish_thresh
        idx1 = 0;
        idx2 = 0;
        idx3 = 0;
        idx4 = 0;
        done = 1;
        break;
    end

    % Find first peak in peak-group
    if ((outRIR1(idx) > thresh) ||
        (outRIR2(idx) > thresh) ||
        (outRIR3(idx) > thresh) ||
        (outRIR4(idx) > thresh))
```

```
idx1 = idx;
idx2 = idx;
idx3 = idx;
idx4 = idx;

% Find individual peaks after first peak
while outRIR1(idx1) <= thresh
    idx1 = idx1 + 1;
end
while outRIR2(idx2) <= thresh
    idx2 = idx2 + 1;
end
while outRIR3(idx3) <= thresh
    idx3 = idx3 + 1;
end
while outRIR4(idx4) <= thresh
    idx4 = idx4 + 1;
end

% Check if these peaks are all valid
[min_v, min_idx] = min([idx1, idx2,
        idx3, idx4]);
[max_v, max_idx] = max([idx1, idx2,
        idx3, idx4]);
if max_v - min_v < array_thresh
    % Valid - clear out peaks so
    % we don't see them later
    outRIR1(idx1) = 0;
    outRIR2(idx2) = 0;
    outRIR3(idx3) = 0;
    outRIR4(idx4) = 0;
    break;
else
    % Clear out starting peak to
    % skip over it
    switch min_idx
```

```
                case 1
                    outRIR1(idx1) = 0;
                case 2
                    outRIR2(idx2) = 0;
                case 3
                    outRIR3(idx3) = 0;
                case 4
                    outRIR4(idx4) = 0;
                otherwise
                    warning('find_peak_group
                        error');
            end
            continue;
        end
    end
    % Go to next index
    idx = idx + 1;
end

end
```

```
function [v_x, v_y] = compute_virtual_source(idx1, idx2,
                                             idx3, idx4);
%COMPUTE_VIRTUAL_SOURCE Computes the location of the
%                       virtual source using the time
%                       delays extracted from each
%                       microphone's RIR of the
%                       tetrahedral microphone array
%
%   [v_x, v_y] = COMPUTE_VIRTUAL_SOURCE(idx1, idx2,
%                    idx3, idx4) calculates the x and y
%           coordinates of the virtual source using the
%           time delays of a peak group extracted from
%           each microphone's RIR of the tetrahedral
%           microphone array
%
%   idx1    = time delay of a peak in the current peak
%             group taken from microphone 1 (in samples)
%   idx2    = time delay of a peak in the current peak
%             group taken from microphone 2 (in samples)
%   idx3    = time delay of a peak in the current peak
%             group taken from microphone 3 (in samples)
%   idx4    = time delay of a peak in the current peak
%             group taken from microphone 4 (in samples)
%
%   Created by: Christopher Co
%
%   Copyright 2012 Christopher Co


% Initialize variables
c = 343;            % Speed of sound
freq = 44100;       % Sampling frequency
a = 0.1500;         % Length of a side of a
                    %   tetrahedral microphone array
```

```
% Convert from peak time delays to distances
r1 = (idx1/freq) * c;
r2 = (idx2/freq) * c;
r3 = (idx3/freq) * c;
r4 = (idx4/freq) * c;


% Spherical wave model equations
d21 = r2 - r1;
d31 = r3 - r1;
d41 = r4 - r1;

v_y = (-2*r1*(d31 - d41) + d31^2 - d41^2)/(2*a);
v_x = (2*r1*(d41 + d31 - 2*d21) + d41^2 + d31^2
            - 2*d21^2)/(-2*sqrt(3)*a);
v_z = (2*d21*r1 + d21^2 - 2*sqrt(3)/3*a*v_x
            + 1/3*a^2)/(2*sqrt(6)/3*a);
end
```

```
function [w_x, w_y] = compute_wall(v_x, v_y);
% COMPUTE_WALL  Computes the location of the
%               wall/reflector from virtual
%               source information
%
%   [w_x, w_y] = COMPUTE_WALL(v_x, v_y)
%           computes the x and y location of
%           the reflector that created the
%           given virtual source
%
%   v_x     = X location of virtual source
%   v_y     = Y location of virtual source
%
%   Notes:  1.) Assumption is distance between
%               source and microphone array is
%               much smaller compared with
%               distance to wall from array
%
%   Created by: Christopher Co
%
%   Copyright 2012 Christopher Co


% With the assumption, wall location is just
% half the distance between the robot and the
% virtual source location
w_x = v_x/2;
w_y = v_y/2;

end
```

A.4 MLS Functions

```
function out = do_mls_rir(input, rir, normalize)
%DO_MLS_RIR Reconstructs the RIR using the MLS
%           Algorithm
%   [out] = DO_MLS_RIR(input, rir, normalize)
%       sends a periodic MLS signal through the
%       room and circular-cross-correlates the
%       output and periodic input to reconstruct
%       the RIR
%
%   input       = Single MLS input (non-periodic)
%   rir         = Ideal RIR of the room
%   normalize   = Flag to do RIR normalization
%
%   Created by: Christopher Co
%
%   Copyright 2012 Christopher Co


% Create periodic MLS input
p_input=[input,input,input];

% Send periodic MLS input through the room
mls_resp = mls_send(p_input, rir);

% Circular cross-correlation of room response
% and periodic MLS input
p_len = size(p_input);
p_len = p_len(2);
p_h = fcxcorr(mls_resp(1,1:p_len),p_input);

% Extract a single period of the reconstructed
% periodic RIR
len = size(input);
len = len(2);
```

```
h = p_h(1,len+1:len+len);
out = h;

% Normalize RIR if necessary
if normalize
    out = out/max(out);
end

end
```

```
function [sequence]=mls_new(N)
%MLS_NEW    Generates an MLS sequence
%   [sequence] = MLS_NEW(N) generates an MLS
%           sequence of length 2^N - 1
%
%   N   = Shift register length
%
%   Source: http://www.silcom.com/~aludwig/
%       Signal_processing/Maximum_length_sequences.htm


if N == 18; taps=[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1]; end;

if N == 17; taps=[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1]; end;

if N == 16; taps=[0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1]; end;

if N == 15; taps=[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]; end;

if N == 14; taps=[0 0 0 1 0 0 0 1 0 0 0 0 1 1]; end;

if N == 13; taps=[0 0 0 0 0 0 0 0 1 1 0 1 1]; end;

if N == 12; taps=[0 0 0 0 0 1 0 1 0 0 1 1]; end;

if N == 11; taps=[0 0 0 0 0 0 0 0 1 0 1]; end;

if N == 10; taps=[0 0 0 0 0 0 1 0 0 1]; end;

if N == 9; taps=[0 0 0 0 1 0 0 0 1]; end;

if N == 8; taps=[0 0 0 1 1 1 0 1]; end;

if N == 7; taps=[0 0 0 1 0 0 1]; end;

if N == 6; taps=[0 0 0 0 1 1]; end;
```

```
if N == 5; taps=[0 0 1 0 1]; end;

if N == 4; taps=[0 0 1 1]; end;

if N == 3; taps=[0 1 1]; end;

M = 2^N-1;

m = ones(1,N);

regout = zeros(1,M);

for ind = 1:M

buf = mod(sum(taps.*m),2);

m(2:N) = m(1:N-1);

m(1)=buf;

regout(ind) = m(N);

end

comp = ~ regout;
sequence = regout - comp;
```

```
function [mls_out]=mls_send(mls_input, true_RIR)
%MLS_SEND   Sends an MLS signal through a room
%   [mls_out] = MLS_SEND(mls_input, true_RIR) sends
%       mls_input through a room characterized by the
%       true_RIR using convolution and returns the
%       room response.
%
%   mls_input   = Input MLS signal
%   true_RIR    = Ideal RIR of the room
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


mls_out=conv(true_RIR,mls_input);
end
```

```
function [ xc ] = fcxcorr(u1,u2)
%[ xc ] = fcxcorr(u1,u2)
%Uses fft to calculate the circular cross
%correlation of two periodic
%signal vectors.This is equivalent to
%xc(k)=sum(u1.*circshift(u2,k)), but
%much faster, especially for large vectors.
%There is no input checking;
%vectors must be equally sized.
%The result is not normalized.  You can get the
%normalized result using:
% xc_n=fcxcorr(u1,u2)/(norm(u1)*norm(u2));

%copyright Travis Wiens 2009

xc=ifft(fft(u1).*conj(fft(u2)));
```

A.5 Robot Mapping Function

```
function [world_x, world_y] = robot_map(wall_x, wall_y,
        robot_x, robot_y, robot_angle,
        old_world_x, old_world_y)
%ROBOT_MAP  Updates the robot's view of the world with new
%           wall information.
%
%   [world_x, world_y] = ROBOT_MAP(wall_x, wall_y, robot_x,
%            robot_y, robot_angle, old_world_x, old_world_y)
%       adds the wall information from wall coordinates into
%       the robot's known mapping of the world.
%
%   wall_x      = X coordinate of the new wall points (relative
%                 to robot position)
%   wall_y      = Y coordinate of the new wall points (relative
%                 to robot position)
%   robot_x     = X coordinate of the robot (from robot POV)
%   robot_y     = Y coordinate of the robot (from robot POV)
%   robot_angle = Angle of the robot (from robot POV)
%   old_world_x = X coordinates of the currently mapped walls
%                 in the room
%   old_world_y = Y coordinates of the currently mapped walls
%                 in the room
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Figure 3 holds the map the robot uses
figure(3);
hold on;

% Calculate wall locations in relation to robot position
world_x = wall_x*cos(robot_angle*pi/180)-
```

```
            wall_y*sin(robot_angle*pi/180) + robot_x;
world_y = wall_x*sin(robot_angle*pi/180)+
            wall_y*cos(robot_angle*pi/180) + robot_y;

% Plot Robot
plot(robot_x, robot_y, 'ro');

% Plot new walls
plot(world_x, world_y, 'o');

% Plot old walls
if old_world_x == -1000     % -1000 signifies there
                            % are no walls to draw
    % Do nothing
else
    plot(old_world_x, old_world_y, 'ko');

    % Update world map
    world_x = [old_world_x world_x];
    world_y = [old_world_y world_y];
end

end
```

A.6 Robot Navigation Function

```
function [new_robot_x, new_robot_y, new_robot_angle,
    new_g_robot_x, new_g_robot_y, new_g_robot_angle,
    turn_flag] = robot_update(world_x, world_y, robot_x,
    robot_y, robot_angle, wall_x, wall_y, g_robot_x,
    g_robot_y, g_robot_angle, turn_flag)
%ROBOT_UPDATE   Computes the next position the robot
%               will move to
%
%   [new_robot_x, new_robot_y, new_robot_angle,
%   new_g_robot_x, new_g_robot_y, new_g_robot_angle,
%   turn_flag] = ROBOT_UPDATE(world_x, world_y, robot_x,
%           robot_y, robot_angle, wall_x, wall_y,
%           g_robot_x, g_robot_y, g_robot_angle,
%           turn_flag)
%   computes the next robot locations based on the
%   current wall information. The decision is based on
%   the right-wall-follow algorithm where the robot finds
%   a wall, follows along that wall on its right side,
%   and turns as necessary.
%
%   world_x         = X coordinates of the robot's current
%                       map of the walls in the room
%   world_y         = Y coordinates of the robot's current
%                       map of the walls in the room
%   robot_x         = X coordinate of the robot's current
%                       position (from robot's POV)
%   robot_y         = Y coordinate of the robot's current
%                       position (from robot's POV)
%   robot_angle     = Angle of robot (from robot's POV).
%                       In degrees.
%   wall_x          = X coordinates of the robot's
%                       currently detected walls
%   wall_y          = Y coordinates of the robot's
%                       currently detected walls
```

```
%   g_robot_x       = Global X coordinate of the robot's
%                       current position (from world POV)
%   g_robot_y       = Global Y coordiante of the robot's
%                       current position (from world POV)
%   g_robot_angle   = Global robot angle (from world POV)
%   turn_flag       = Flag to detect turns
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Initialize Variables
move = 0.2;             % Maximum we are allowing the
                        %   robot to move (in meters)

search_x_dist = 0.3;    % Search window for walls
search_y_dist = 0.3;

desired_x_dist = 1;     % Desired distances for walls
desired_y_dist = 1;

turn_thresh = 1.0;      % Distance before turn (in meters)

turning_flag = 0;       % Flag that signifies if we will
                        %   need to turn in this iteration

K_p_x = 0.1;            % Controller gains
K_p_y = 0.1;

num_walls = size(wall_x);
num_walls = num_walls(2);

% Current Assumption - only one data point for right
% wall.
for i = 1:num_walls
```

```
    % Search for data points corresponding with
    % right walls
    if wall_y(i) < 0
        if (wall_x(i) > -search_x_dist) &&
            (wall_x(i) < search_x_dist)
            right_wall_x = wall_x(i);
            right_wall_y = -wall_y(i);
        end
    end
    % Search for data points corresponding with
    % front walls
    if wall_x(i) > 0
        if (wall_y(i) > -search_y_dist) &&
            (wall_y(i) < search_y_dist)
            front_wall_x = wall_x(i);
            front_wall_y = wall_y(i);
        end
    end
end


% If we need to turn, set turning_flag
if exist('front_wall_x', 'var') == 0
    front_wall_x = desired_x_dist;
    front_wall_y = desired_y_dist;
else
    if front_wall_x < turn_thresh
        turning_flag = 1;
    end
end


front_wall_x = desired_x_dist;
front_wall_y = desired_y_dist;
if exist('right_wall_x', 'var') == 0
    right_wall_x = 0;
```

```
    right_wall_y = desired_y_dist;
end

% Compute errors
error_x = right_wall_x - desired_x_dist;
error_y = right_wall_y - desired_y_dist;

% Apply P controller gains
force_p_x = -K_p_x * error_x;
force_p_y = -K_p_y * error_y;

% Compute overall force to apply to plant
force_x = force_p_x;
force_y = force_p_y;

% Compute new movement based on forces applied
new_robot_mag = sqrt((force_x)^2+(force_y)^2);
new_robot_angle = atan2(force_y,force_x)*180/pi;

if new_robot_mag > move
    new_robot_mag = move;
end

if turning_flag == 1
    turn_flag = turn_flag+1;
    if turn_flag == 4
        turn_flag = 0;
    end
    turning_flag = 0;
end

if turn_flag == 1
    new_robot_angle = new_robot_angle + 90;
elseif turn_flag == 2
        new_robot_angle = new_robot_angle + 180;
elseif turn_flag == 3
```

```
    new_robot_angle = new_robot_angle + 270;
end

%Update all output values
new_robot_x = robot_x +
        new_robot_mag*cos(new_robot_angle*pi/180);
new_robot_y = robot_y +
        new_robot_mag*sin(new_robot_angle*pi/180);

new_g_robot_angle = g_robot_angle +
        (new_robot_angle - robot_angle);
new_g_robot_x = g_robot_x +
        new_robot_mag*cos(new_g_robot_angle*pi/180);
new_g_robot_y = g_robot_y +
        new_robot_mag*sin(new_g_robot_angle*pi/180);
end
```

A.7 Error Calculation Function

```
function [sum_error, count] = compute_error(w_x, w_y, r_x, r_y)
%COMPUTE_ERROR  Computes the total squared error of all sampled
%               data points and their actual wall locations for
%               calculation of average RMS error
%
%   [sum_error, count] = COMPUTE_ERROR(w_x, w_y, r_x, r_y)
%       computes the sum of the squared error of the sampled
%       wall points and the actual locations of the wall.
%       This function is used to aid in the calculation of
%       the average RMS error.  In addition, the total number
%       of sampled data points is returned in count.
%
%       The function computes the RMS error against all the
%       walls and picks the lowest RMS error to correlate
%       that specific data point to a specific wall.
%
%   w_x     = X coordinates of sampled wall location data
%   w_y     = Y coordinates of sampled wall location data
%   r_x     = X location of robot from world POV
%   r_y     = Y location of robot from world POV
%
%   Notes:  1.) This function is hard-coded to evaluate
%               sampled wall data against a 10m by 10m
%               by 10m room.
%           2.) Walls are assumed to be either vertical
%               or horizontal with respect to the world POV
%
%   Created by: Christopher Co
%
%   Copyright 2012 Christopher Co


% Initialize variables
num_wall = size(w_x);
```

```
num_wall = num_wall(2);     % Number of wall points
sum_error = 0;              % Sum of squared errors
x_min = 0;                  % Left wall location in world POV
x_max = 10;                 % Right wall location in world POV
y_min = 0;                  % Lower wall location in world POV
y_max = 10;                 % Upper wall location in world POV


% Compute difference between robot's current position (in
% world POV) and the wall locations to translate the wall
% locations into the robot's POV
trans_l = x_min - r_x;
trans_r = x_max - r_x;
trans_d = y_min - r_y;
trans_u = y_max - r_y;


% Loop over every sampled wall data point
for i = 1:num_wall

    % Compute the squared difference between sampled data
    % points and actual wall locations for each major
    % direction
    diff_l = (w_x(i) - trans_l)^2;
    diff_r = (w_x(i) - trans_r)^2;
    diff_u = (w_y(i) - trans_u)^2;
    diff_d = (w_y(i) - trans_d)^2;

    % Use the lowest squared difference and add to total
    % squared error (Assumption: the wall direction with
    % the lowest squared difference is the wall the
    % sampled data point originated from)
    diff = [diff_l, diff_r, diff_u, diff_d];
    sum_error = sum_error + min(diff);
end
```

```
% Return total number of sampled data points (useful
% for calcualating average RMS error)
count = num_wall;
end
```

APPENDIX B

QUADRATIC INTERPOLATION CODE

B.1 Execute Script

```
%EXECUTE Execute Simulation Script
%   This Matlab script executes the quadratic
%   interpolation optimization simulation and
%   computes the associated errors that are
%   reported in the thesis.
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Create Room/RIRs

%% Base Room with 1 virtual source
% fs = 44100;
% mic=[2 2 2];
% n = 1;
% r = 0.2;
% rm = [4 4 4];
% src = [3 2 2];
% h_list = rir(fs, mic, n, r, rm, src);

% Advanced Room
fs = 44100;
mic=[2 2 2];
```

```
n = 4;
r = 0.2;
rm = [4 4 4];
src = [3 2 2];
h_list = rir(fs, mic, n, r, rm, src);


% Calculate the baseline error in the true RIR
% from rounding to the nearest sample
rounded = round(h_list(:,2));
rounderror = calc_error(h_list(:,2),rounded);


% Do Petersen sinc interpolation
samp_h = sinc_interp(h_list);


% Ideal RIR Quadratic Interpolation %

% Loop until we explore all of the peaks
currenth = samp_h;
indices = 0;
num_peak = 0;
while 1
    % Find the next peak
    [currenth, indices] = find_one_peak(currenth);

    % Check if finished
    if indices == -1
        break;
    end

    % Apply quadratic fitting
    fit_coeff = quad_fit(samp_h, indices);
    num_peak = num_peak+1;
    cur_peak = indices(1);
```

```
    % Find the sample the peak of the
    % quadratic fit curve and record that
    % as our "peak" value
    peaks(num_peak) = cur_peak -
        fit_coeff(2)/(2*fit_coeff(1));
end

% Compute error between interpolated peak
% locations and the true peak locations
peaks = sort(peaks);
peaks = peaks';
error = calc_error(h_list(:,2),peaks);


% MLS Reconstructed RIR Quadratic Interpolation %

% Generate MLS input signal
mls_num = 15;
mls_input = mls_new(mls_num);

% Simulate the sending of MLS input through room
mls_resp = mls_send(mls_input, samp_h);

% Compute the estimated impulse response
% from the MLS signal
mls_cross = xcorr(mls_resp,mls_input);
mls_cross_size = size(mls_cross);
mls_cross_size = mls_cross_size(2);
half_mls_cross_size = ceil(mls_cross_size/2);
mls_trunc_cross =
    mls_cross(half_mls_cross_size:mls_cross_size);
out = mls_trunc_cross;

% Normalize reconstructed RIR
normout = out/max(out);
```

```
% Generate Robot's RIR
thresh_mls = normout;
thresh_size = size(mls_input);
thresh_size = thresh_size(2);

thresh_mls = thresh_mls(1:thresh_size);
thresh_mls = thresh_mls';
currenth = thresh_mls;

% Loop until we explore all RIR peaks
indices = 0;
num_peak = 0;
while 1
    % Find the next peak
    [currenth, indices] = find_one_peak(currenth);

    % Check if we are done
    if indices == -1
        break;
    end

    % Apply quadratic fitting
    fit_coeff = quad_fit(thresh_mls, indices);
    num_peak = num_peak+1;
    cur_peak = indices(1);
    % Find the sample the peak of the
    % quadratic fit curve and record that
    % as our "peak" value
    mls_peaks(num_peak) = cur_peak -
                fit_coeff(2)/(2*fit_coeff(1));
end

% Compute error between interpolated peak
% locations and the true peak locations
mls_peaks = sort(mls_peaks);
mls_peaks = mls_peaks';
```

```
mls_error = calc_error(h_list(:,2),mls_peaks);

% Compute error between interpolated peak
% locations rounded to the nearest sample
% and the true peak locations
mls_rounded = round(mls_peaks);
mls_rounderror = calc_error(h_list(:,2),
                        mls_rounded);
```

B.2 Sinc Interpolation Function

```
function [out] = sinc_interp(hlist)
%SINC_INTERP    Apply Petersen sinc
%               interpolation to RIR
%
%   [out] = SINC_INTERP(hlist)
%
%   hlist   = List of true RIR peak indices
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Create output array
out = zeros(round(max(hlist(:,2)))+10,1);

num = size(hlist);
num = num(1);

% For each peak...
for i = 1:num

    % Compute the values of the sinc
    % whose peak is centered at the
    % peak for a window of [-2,2]
    % around the peak sample
    for j = -2:2
        current = round(hlist(i,2))+j;
        temp = hlist(i,1)*sinc(3/5*
                (current-hlist(i,2)));
        if temp > 0
            % Have to check if we get
            % overlapping impulses.
            % Check if sample value
```

```
            % is non-zero
            out(current) = temp;
        end
    end
end
end
```

B.3 Peak Picking Function

```
function [newh, indices] = find_one_peak(h)
%FIND_ONE_PEAK  Finds the first peak in h
%
%   [newh, indices] = FIND_ONE_PEAK(h)
%       finds the first peak in h (RIR) that
%       is above a specified threshold.
%       When found, that index is returned
%       to the calling function and a new
%       RIR with that peak zeroed out is
%       returned so that on subsequent
%       calls to this function, the next
%       peak will be obtained.
%
%   h   = RIR to search for first peak
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Initialize Variables
newh = h;

% Loop until we finish
while 1
    [p_val, p_i] = max(newh);
    threshold = 0.015;
    samp_thresh = 0.002;
    % Check if no samples are left...
    if p_val < threshold
        indices = -1;
        return;
    end
```

```
% Process argmax peak
size = 1;
cur_i = p_i;
% Search left
while 1
    % Check if test sample is greater
    % than sample threshold
    if h(cur_i-1) < samp_thresh
        break
    end
    % Check if test sample is lower
    % than current sample
    if h(cur_i) <= h(cur_i-1)
        break
    end
    % Move window left and increment
    % total size
    newh(cur_i) = 0;
    cur_i = cur_i - 1;
    size = size + 1;
end

% Save left window and reload current
% sample with peak value
newh(cur_i)=0;
left_i = cur_i;
cur_i = p_i;

% Search right
while 1
    % Check if test sample is greater
    % than sample threshold
    if h(cur_i+1) < samp_thresh
        break
    end
    % Check if test sample is greater
```

```
        % than current sample
        if h(cur_i) <= h(cur_i+1)
            break
        end
        % Move window right and increment
        % total size
        newh(cur_i) = 0;
        cur_i = cur_i + 1;
        size = size + 1;
    end
    newh(cur_i) = 0;
    right_i = cur_i;

    % Check size is good for quadratic
    % fit...(>3)
    if size >= 3
        break
    end

    str = sprintf('Not enough info at
                   Peak %d',cur_i);
    disp(str);
end
indices = left_i:1:right_i;

end
```

B.4 Quadratic Interpolation Function

```
function [out] = quad_fit(h, indices)
%QUAD_FIT   Calculate quadratic fit
%           curve coefficients
%
%   [out] = QUAD_FIT(h, indices)
%       obtains the points to apply
%       the quadratic fit against,
%       then computes the coefficients
%       of the quadratic fit and
%       returns those values.  Those
%       coefficients can then be used
%       to find the peak value of
%       the quadratic curve
%
%   h       = RIR
%   indices = Sample numbers to apply
%               quadratic fit to
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Get data points
y = h(indices);
x = (0:size(y)-1)';

% Compute quadratic fit coefficients
A = [x.^2 x ones(size(x))];
out = (A'*A)\(A'*y);

end
```

B.5 Error Calculation Function

```
function error = calc_error(h_list, peaks)
%CALC_ERROR Calculates the error between
%           the peaks in the first argument
%           and the peaks in the second
%           argument
%
%   error = CALC_ERROR(h_list, peaks)
%       walks through the peak arrays passed in
%       both arguments, and does a pairwise
%       comparison (assuming difference is within
%       a specific threshold value), and computes
%       the average RMS error between the two
%       arrays
%
%   h_list  = List of RIR samples where true
%               RIR peaks occur
%   peaks   = List of RIR samples where
%               sampled/interpolated RIR peaks
%               occur
%
%   Created by: Christopher Co
%
%   Copyright  2012 Christopher Co


% Initialize variables

error = 0;
thresh = 2;
num = size(h_list);
num_true = num(1);
num = size(peaks);
num_exp = num(1);
j = 1;                  % Index for walking through
```

```
                        %   true RIR list


% Loop across all recorded peak samples
for i = 1:num_exp       % Index for walking through
                        %   interp RIR list

    % Check if the current peaks correspond to
    % each other
    if abs(h_list(j)-peaks(i)) < thresh

        % If so, add to total squared error and
        % move to next peaks
        error = error + (h_list(j) - peaks(i))^2;
        j = j+1;
    else
        % Search for the corresponding peak
        % (spurious or missing peak)
        testj = j;
        found = 0;

        % Search until we reach the end of list
        while testj ~= num_true
            if abs(h_list(testj)-peaks(i)) < 2
                % Found!  Add to total squared error
                found = 1;
                j = testj;
                error = error + (h_list(j) - peaks(i))^2;
                j = j+1;
                break;
            else
                % Missed true peak in interp RIR
                str = sprintf('Missed Peak at %d', h_list(testj));
                disp(str);
                testj = testj+1;
            end
```

```
        end

        % Check if we found the peak in our search
        if found == 1
            continue
        else
            % Spurious peak in interp RIR
            str = sprintf('Spurious Peak at %d', peaks(i));
            disp(str);
            j = j+1;
            continue
        end
    end
end

% After we exhaust interp RIR peak list, any
% remaining true RIR peaks in the list were
% missed in interp RIR peak list
while j <= num_true
    str = sprintf('Missed Peak at %d', h_list(j));
    disp(str);
    j = j+1;
end

% Compute average RMS error
error = error/num_exp;
error = sqrt(error);
end
```

REFERENCES

[1] J. B. Kobler, B. S. Wilson, O. W. H. Jr., and A. L. Bishop, "Echo intensity compensation by echolocating bats," *Hearing Research*, vol. 20, no. 2, pp. 99–108, 1985.

[2] G. Jones, "Echolocation," *Current Biology*, vol. 15, no. 13, pp. R484–R488, 2005.

[3] J. H. Lim, J. Leonard, and S. K. Kang, "Mobile robot relocation using echolocation constraints," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 1999, pp. 154–159.

[4] A. Kurz, "ALEF: An autonomous vehicle which learns basic skills and constructs maps for navigation," *Robotics and Autonomous Systems*, vol. 14, no. 2-3, pp. 171–183, 1995.

[5] A. Ohya, Y. Nagashima, and S.-I. Yuta, "Exploring unknown environment and map construction using ultrasonic sensing of normal direction of walls," in *IEEE Proc. International Conference on Robotics and Automation*, May 1994, pp. 485–492.

[6] H. Atmoko, D. C. Tan, G. Y. Tian, and B. Fazenda, "Accurate sound source localization in a reverberant environment using multiple acoustic sensors," *Measurement Science and Technology*, vol. 19, no. 2, pp. 1–10, 2008.

[7] J. Valin, F. Michaud, J. Rouat, and D. Letourneau, "Robust sound source localization using a microphone array on a mobile robot," in *Proc. IEEE Intelligent Robots and Systems (IROS'03)*, Las Vegas, Nevada, 2003, pp. 1228–1233.

[8] J.-S. Hu, C.-H. Yang, and C.-K. Wang, "Sound source localization by microphone array on a mobile robot using eigen-structure based generalized cross correlation," in *IEEE Workshop on Advanced Robotics and Its Social Impacts*, Aug. 2008, pp. 1–6.

[9] Q. Xu, P. Yang, J. Wang, and H. Sun, "Sound source localization system based on mobile robot," in *24th Chinese Control and Decision Conference (CCDC)*, May 2012, pp. 204–207.

[10] B. Gunel, "Room shape and size estimation using directional impulse response measurements," in *Proc. Forum Acusticum*, Sep. 2002.

[11] E. A. P. Habets, "Room impulse response generator," Technische Universiteit Eindhoven, Tech. Rep., 2006.

[12] M. Sondhi, D. Morgan, and J. Hall, "Stereophonic acoustic echo cancellation — An overview of the fundamental problem," *IEEE Signal Processing Letters*, vol. 2, no. 8, pp. 148–151, Aug. 1995.

[13] W. G. Gardner, B. L. Vercoe, and S. A. Benton, "The virtual acoustic room," 1992.

[14] G.-B. Stan, J.-J. Embrechts, and D. Archambeau, "Comparison of different impulse response measurement techniques," *J. Audio Eng. Soc*, vol. 50, no. 4, pp. 249–262, 2002.

[15] M. R. Schroeder, "New method of measuring reverberation time," *The Journal of the Acoustical Society of America*, vol. 37, no. 6, pp. 1187–1188, 1965.

[16] J. Vanderkooy, "Aspects of mls measuring systems," *J. Audio Eng. Soc*, vol. 42, no. 4, pp. 219–231, 1994.

[17] D. D. Rife and J. Vanderkooy, "Transfer-function measurement with maximum-length sequences," *J. Audio Eng. Soc*, vol. 37, no. 6, pp. 419–444, 1989.

[18] M. Cohn and A. Lempel, "On fast m-sequence transforms," *IEEE Transactions on Information Theory*, vol. 23, no. 1, pp. 135–137, Jan. 1977.

[19] J. B. Allen and D. A. Berkley, "Image method for efficiently simulating small-room acoustics," *The Journal of the Acoustical Society of America*, vol. 65, no. 4, pp. 943–950, 1979.

[20] P. Peterson, "Simulatng the response of multiple microphones to a single acoustic source in a reverberant room," *J. Acoust. Soc. Am.*, vol. 80, pp. 1527–1529, 1986.

[21] A. Beck, P. Stoica, and J. Li, "Exact and approximate solutions of source localization problems," *IEEE Transactions on Signal Processing*, vol. 56, no. 5, pp. 1770–1778, May 2008.

[22] Y. Huang, J. Benesty, G. Elko, and R. Mersereati, "Real-time passive source localization: A practical linear-correction least-squares approach," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 8, pp. 943–956, Nov. 2001.

[23] H. Schau and A. Robinson, "Passive source localization employing intersecting spherical surfaces from time-of-arrival differences," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 8, pp. 1223–1225, Aug. 1987.

[24] X. Lai and H. Torp, "Interpolation methods for time-delay estimation using cross-correlation method for blood velocity measurement," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 46, no. 2, pp. 277–290, Mar. 1999.

[25] S. McGovern, "A model for room acoustics," 2004. [Online]. Available: http://sgm-audio.com/research/rir/rir.html