2021-12-20

Development and Characterization of an Underwater Acoustics Laboratory Via in situ Impedance Boundary Measurements

Cameron Taylor Vongsawad
Brigham Young University

Development and Characterization of an Underwater Acoustics Laboratory

Via *in situ* Impedance Boundary Measurements

Cameron Taylor Vongsawad

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Development and Characterization of an Underwater Acoustics Laboratory
Via *in situ* Impedance Boundary Measurements

Cameron Taylor Vongsawad
Department of Physics and Astronomy, BYU
Master of Science

Modeling underwater acoustic propagation comes with a variety of challenges due to the need for proper characterization of the environmental conditions. These conditions include changing and complex water properties as well as boundary conditions. The BYU underwater acoustics open-air tank test-bed and measurement chain were developed to study underwater acoustic propagation within a controlled environment. This lab was also developed to provide ways to test and validate ocean acoustics models and machine learning algorithms without the high cost associated with obtaining open-ocean measurements. However, tank measurements require characterization of boundary conditions associated with the walls of the tank which create lateral reflections not present in an open ocean. The characterization of BYU's underwater acoustic tank included measuring the calibrated impulse response of the tank through frequency deconvolution of swept-sine signals to determine the frequency-dependent reverberation time through reverse Schroeder integration. The reverberation time allows for calculating the frequency dependent spatially averaged acoustic absorption coefficient of the tank enclosure boundaries using methods common to room acoustics and also yield insights into the Schroeder frequency limit of the tank. Time-of-arrival measurements are used to validate models for quantifying the speed of sound in the water. The acoustic characterization was validated by comparison with predicted values and also applied to measurements in the tank lined with anechoic panels to reduce lateral reflections. An initial investigation into effective tank models evaluated the idealized rigid-wall and pressure-release water-air boundary model, a finite-impedance boundary model applying the measured acoustic boundary absorption and a benchmark open ocean model, known as ORCA, to determine potential tank model candidates. This study demonstrates the efficacy of the methodology for underwater acoustic tank characterization, provides a frequency dependent acoustic boundary evaluation from 5-500 kHz, and provides an initial comparison of tank models with applied characterization.

Keywords: acoustic, anechoic panels, boundary absorption, characterization, deconvolution, finite-impedance boundary, *in situ* calibration, lab design, normal-mode waveguide modeling, reverberation time, Schroeder frequency, through the sensor, ultrasonic acoustic propagation, underwater acoustics, water tank

ACKNOWLEDGMENTS

Once upon a time I could not have imagined myself where I am today. My experience as both an undergraduate student and again as a graduate student at BYU has played a major role in helping me become the man I am today. And I really like who I am today.

I greatly attribute much of my educational success to the opportunities provided to me by my advisor Dr. Tracianne Neilsen, who has opened doors for my personal and professional growth since 2013. She put a lot of faith in me when she asked me to come back to BYU and help her build the underwater acoustics laboratory and start an experimental research group. She has been the most supportive mentor who has helped me understand that my potential rises well above the limits I once imposed on myself and supported me sincerely the whole way. I am glad I made the decision to come back to BYU and work with her again. She is seriously one of the coolest people I know.

Duane Merrell, Timothy Leishman, Brian Anderson, Scott Sommerfeldt, Kent Gee, Adam Bennion and the rest of the faculty in the department of Physics and Astronomy at BYU have been inspiring both intellectually and spiritually throughout my journey. During two degrees, all the faculty I have interacted with have pushed me and given me the opportunity to grow and become a better person every day. I am proud to say that I have been educated by them.

I am thankful for fellow graduate students Adam Kingsley and Ian Bacon who have been great research supports, office mates, classmates and friends in this journey. I am also thankful for Gabriel Fronk, Kaylyn Terry, Corey Dobbs, and Scott Hollingsworth for being members of the greatest research team I could ask for. My peers helped coming to class, doing homework, and performing research be an awesome experience even on the most frustrating of days when nothing seemed to be working or make any sense.

Finally, I am thankful for funding from the Office of Naval Research, Defense University Research Instrumentation Program N00014-18-S-F007 Grant 12671398 which has made all of this possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Sound navigation ranging (SONAR) has evolved much since its development during World War II. Safely navigating the ocean environment, obstacles, other submarines, and ships remains a difficult task for submarine SONAR technicians who strive to see through hearing. Analysis of what is heard through the multitude of on-board sensors has inherent uncertainty. Active SONAR relies on listening to echoes of pings. The time delays between the echoes allow one to compute the direction and distance from an object, with limited information about the environment, range and depth.

Active SONAR is also used to scan an area of the seafloor and map out reflective interfaces in the seafloor. One method is through the use of side-scan SONAR on a research vessel. A conical or fan-shaped beam is swept across the seafloor as the ship moves along different paths. The echoes collected by the side-scan SONAR can then be used to estimate the structure of the seafloor. A schematic of this process using a towed side-scan SONAR array is shown in Fig. 1.1 [1]. The depth at which the side-scan SONAR penetrates the seafloor depends on the frequency and level of the signals used. Typically side-scan SONAR is very loud and time consuming, so only small portions

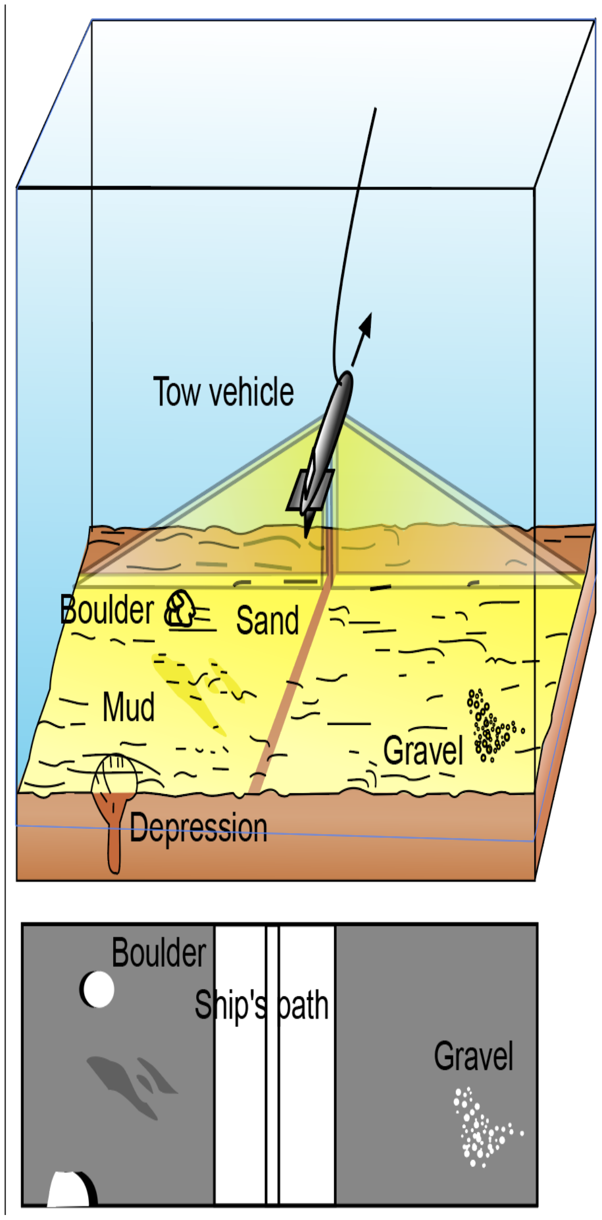


Figure 1.1 Illustration of how side-scan SONAR from a towed side-scan SONAR array works. The conical or fan-shaped beams are directed to the seafloor and the echoes are processed to identify the types of materials. Retrieved from Wikimedia commons.

of the seafloor can be mapped in this manner. Another method uses an echo sounder for sub-bottom profiling, but again the time and costs involved limit the amount of the seafloor that can be scanned.

Although active SONAR is one way to use sound to probe the ocean environment, the required high sound levels do not allow this to be done with any degree of stealth. The stealthy way to use sound in the ocean is called passive SONAR. Passive SONAR seeks to "see" by only listening to ocean sounds. Passive SONAR can be used for detecting, locating, and tracking sound sources and used to estimate properties of the ocean environment. Advantages to passive SONAR include increased stealth and reducing overall noise pollution in the ocean [2].

The challenges of passive SONAR center on significant refraction as the sound propagates in the water and complicated interactions with the seabed. The challenges are different based on two major categories: shallow ocean environments (depth < 200m, encompassing about 5% of the world's oceans) and deep ocean environments (depth > 200m). In either realm of ocean environments, Snell's law dictates that sound paths in the water vary significantly due to depth-dependent changes in the temperature, salinity, and ambient pressure that influence the speed of sound [3]. Shallow ocean environments especially experience dramatic variations due to warming and cooling temperatures, which shift by latitude, season, or current weather conditions [4]. This variability leads to a depth-dependent temperature gradient which means sound speed profiles differ in specific environments for underwater acoustics. Further complications with sound speed arise with refraction, absorption, and reflection effects caused by the complicated boundary conditions due to characteristics of seabed layering. Because these environmental properties affect sound propagation, uncertainty in environment properties make it difficult to localize and classify sources of sound. These refractive effects and other complexities of an ocean environment complicate modeling underwater sound propagation in the shallow ocean and the detection, localization, and tracking of sound sources.

The field of work that concentrates on using passive SONAR to determine seafloor properties is called geoacoustic inversion. Geoacoustic inversions are based on a model of the sound propagation, i.e., a computer model that numerically solves the the wave equation accounting for refraction, reflection, and transmission. The models require a set of parameters that contain acoustically relevant properties of the environment. An acousto-elastic parameterization is one where each depth is defined by compressional sound speed, compressional attenuation and density. Most of the commonly used open-ocean models are formulated in cylindrical coordinates (range, depth, and angle) with assumed azimuthal symmetry (no angular dependence), which makes them two-dimensional models: modeling the sound propagation in a vertical-horizontal plane. Some of the common approaches to modeling include ray-tracing, normal-mode formulation, parabolic equation solvers, and wavenumber integration. Many of these models are open-source and are available at the Ocean Acoustics Library (https://oalib-acoustics.org/).

Optimal ways to use passive SONAR and employ geoacoustic inversions have been researched for many decades. Increased computational capacity and advanced algorithms developed over the past decade, in particular, have now made it possible to pursue a different approach to source detection, localization, and tracking, and estimating properties of the ocean environment through the use of machine learning. The primary advantage to deploying machine learning in ocean acoustics is the potential for real-time applications. Traditional geoacoustic inversions require a tremendous amount of modeling across the parameter space (often through the use of Markov Chain Monte Carlo Sampling) as the model-data mismatch is minimized for each new data sample before an answer can be obtained. With machine learning, the computationally extensive modeling and training can be completed once before an experiment or mission begins. The trained model can then be applied to each measured sample to obtain real-time predictions.

While machine learning algorithms are already being used in many fields and aspects of our daily lives, applications to ocean acoustics are complicated by one main problem: the lack of labeled field data. Most machine learning for computer vision, speech processing, and other tasks is accomplished by supervised training. Supervised learning requires the training data be labeled with the correct values the algorithm is trying to learn to predict. In ocean acoustics, the plethora of data recorded over the decades is not labeled. In particular, the correct parameterization of the seafloor at the location the data were recorded is not known. The lack of labeled data in ocean acoustics has led BYU's underwater acoustic research group to pursue a supervised machine learning model trained on data simulated with open-ocean models and known parameterizations of the ocean environment.

One potential approach is to train the model on this synthetic data and then refine the model using a small set of labeled measured data. Significant testing is needed to determine how this refinement learning approach can be applied in ocean acoustics. To accommodate this research and further development of machine learning in ocean acoustics, the underwater acoustics lab at BYU was created which can obtain sets of labelled data in a controlled environment for understanding this refinement learning approach.

Efforts to create this lab began in 2019. A Defense University Research Instrumentation Program (DURIP) grant was received to allow for the equipment to be purchased. The work contained in this thesis describes the setup of the lab and initial work to characterize sound propagation in the water tank particularly related to boundary characterization which is fundamental to the development of water tank propagation models. Initial efforts to develop a model that describes the sound propagation in the tank are described, which may be used to obtain synthetic training data and explore refinement learning in the future. This work has laid the foundation for the underwater acoustics lab at BYU to be a valuable research tool for years to come.

1.2 Background

Underwater acoustic propagation is challenging to model due to the need to account for a wide variety of fluid and boundary properties. A laboratory environment can be used to reduce these challenges because it provides opportunities to control experimental conditions. Controlled water tank measurements are commonly used for validating measurement techniques, propagation models, and recently machine learning propagation models [5–11]. However, a controlled tank also brings about challenges not seen in the open-ocean. Many scaled ocean models have been evaluated in tanks [6–8, 12, 13] much smaller or much larger in size than BYU's, and a scaled tank model that can be used for effectively modeling sound propagation in BYU's acrylic water tank is required to continue this work.

Etter [14] gives a comprehensive review of other common open-ocean models, but ORCA will be used as the benchmark for this thesis.ORCA is an open-ocean model [15] considered to provide a good approximation of open-ocean sound propagation [16]. ORCA is a range-independent, normal-mode model that assumes azimuthal symmetry and computes the frequency response of the water waveguide with a depth-dependent sound speed and horizontally stratified seabed. The validity of the frequency response depends on how well the parameterization of the acoustical properties of the environment match reality. ORCA is highly robust accounting for a wide range of real world characteristics such as leaky boundaries (surface and floor) and has low computational costs. ORCA has recently been used by BYU's underwater acoustic research group to effectively simulate data samples for training a Convolutional Neural Network (CNN) for finding seabed type and range for pressure time series data from explosive sources [17] and spectral density levels from a towed tonal source [18], and seabed classification from surface ship spectrograms [19] along with finding ship track parameters [20].

Similar to an open-ocean environment, the rectangular parallelepiped BYU open-air acoustic water tank contains a water waveguide bounded by a pressure-release water-air surface and a floor with a finite impedance boundary. Unlike the open-ocean, the tank also includes acoustically reflective side walls. Time-gating a signal is a common method of applying consideration to wall reflections [12], but this solution is not perfect. To reduce these lateral reflections, anechoic panels can be used along the walls. Part of this work is to assess the efficacy of anechoic panels improving tank models relative to open-ocean models.

The major benefits of tank measurements are the reduction of measurement costs [9,21], as well as improved environmental control [22]. The high economic and temporal costs associated with open-ocean measurements can be reduced with effective tank measurement if the tank can function as an effective scaled model of ocean sound propagation. Further, the tank allows for control of environmental parameters that greatly affect the sound speed of the water such as temperature, salinity and seabed type. This aspect may allow evaluating the above mentioned CNN's ability to perform under sound speed variability. Our lab's goal is to use the tank to improve understanding of acoustic ocean propagation.

The purpose of this thesis is to characterize the tank boundaries in order to evaluate common models for effectively modeling acoustic propagation in the tank. Many tanks have been characterized using the reverberation method [10,23–26], but few have applied this estimated absorption to model sound propagation in the tank. Understanding the acoustic boundary conditions of the tank is necessary for developing a more precise modal propagation model. With measured characterization of the frequency-dependent absorption of the tank boundaries, a finite-impedance boundary model may be used instead of an idealized rigid-boundary, pressure-release boundary, or assumed impedance boundary model which may allow for the evaluation and development of an underwater acoustic measurement tank.

The classic reverberation method used in room acoustics has been modified to evaluate the additional accounting of propagation absorption. This characterization of the tank may be performed *in situ* to model the tank with acrylic boundaries, partial anechoic boundaries, as well as any other boundary such as artificial seabeds to be applied in the future. An effective tank model, especially when anechoic paneling is applied to significantly reduce side wall reflections, can lead to the development of improved open-ocean modeling capabilities within a tank environment.

1.3 Thesis Overview

This thesis presents research characterizing BYU's underwater acoustic measurement tank and provides initial analyses of the effectiveness of various tank models. These tank models and the open-ocean model ORCA are compared to responses obtained from measurements with and without the anechoic panels to determine if the side wall reflections are sufficiently reduced that the tank can act as a scale-mode ocean. for modeling a scaled open-ocean relative to the ORCA model. Chapter 2 provides a look into how the experimental setup has been developed from the ground up for general underwater acoustic measurements in the tank. Chapter 3 details the measurement and calibration process, the theory of data processing for acoustic characterization, as well as an overview of the models which require acoustic characterization and their potential benefits for this research. Chapter 4 details the experimental results evaluating the boundary characterization of the water tank and investigates how applying these boundary conditions to various models influences model-measures comparisons. Chapter 5 summarizes the main conclusions from the research presented in this thesis, and a summary of additional future work is given. Appendix A details the data collected and used for this thesis for future reference. Appendix B documents the algorithms developed to process the data effectively. Appendix C presents the current laboratory documentation manual for continued student learning and effective transfer of knowledge. This includes a more detailed description of equipment, functionality and maintenance in the laboratory.

Chapter 2

Experimental Setup

2.1 Introduction

The Underwater Acoustics Laboratory at Brigham Young University facilitates student research in underwater acoustics. Each component has been chosen and designed for a high level of safety, automation, and reliability. These features give students the opportunity to learn to perform effective measurements and data analysis. The system opens opportunities for new research in underwater acoustics.

Obtaining large open-water data sets for underwater acoustics research and validating measurements [6, 11, 26] has high economic and temporal costs. A laboratory system saves on those costs [21], especially for researchers without ease of access to large bodies of water [22, 27]. When scaled measurements are acceptable, the water tank is useful to collect large data sets. Open-water tests are often noisy and unpredictable with changing environmental conditions. The tank allows for better control of the environment [22]. Automation allows data to be collected quickly and efficiently, while maintaining high precision.

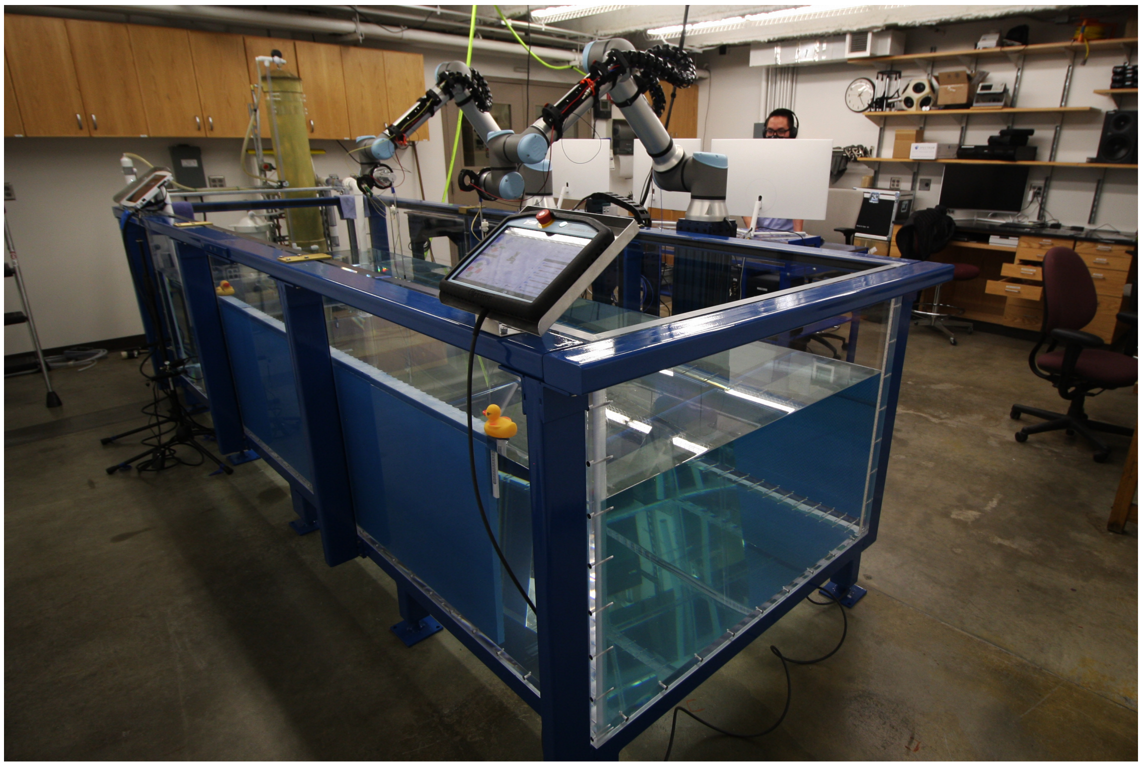


Figure 2.1 BYU's underwater acoustics laboratory with acrylic water tank and robotic positioning system.

This chapter provides details about the tank, signal transmission and recording, automated positioning, and how to validate the measurement chain. A discussion of how this design maintains potential for a wide variety of underwater acoustic laboratory measurements is also given.

2.2 Water Tank

The open-air water tank seen in Fig. 2.1 was made by Engineering Laboratory Design Inc. (Lake City, Minnesota, USA) and is made of scratch resistant acrylic panels, solvent welded together, with a steel frame on adjustable leveling pads. The tank material, acrylic, was chosen for its visual transparency and non-corrosive nature. Acrylic also has an acoustic impedance which is closer to that of water, thus, reducing reflections more than common tank materials such as steel, concrete, iron or glass [21, 23, 28, 29].

The side walls can be lined with attenuating material as seen in Fig. 2.3 (see Fig. 2.2 for comparison with the tank without panels) to further reduce acoustic reflections and the reverberation time. The attenuating material from Precision Acoustics was chosen to reduce side-wall reflections especially for ultrasonic frequencies. The 50 mm thick, 60 cm tall, square Apltile SF5048 panels advertise an echo reduction greater than 30 dB for the recommended frequency band of 20-200 kHz. This reduction improves the ability to model an open-water environment within the tank by avoiding extra unwanted superposition due to side wall reflections. An investigation into the broadband response of the panels is discussed in Sec. 3.1.4, and the results of this investigation are given in Sec. 4.3.

The tank's dimensions were chosen to allow scaled acoustical measurements, similar to those described in Refs. [6–8, 21, 26, 30] and designed to be large enough for varied applications as well as maximize usable laboratory space. The 3.66 m long by 1.22 m wide rectangular tank has a maximum water depth of 0.92 m, corresponding to a maximum fill volume of 4077.6 L.

A valve is located in one corner of the bottom sheet of acrylic to allow ease of both draining and filling without the mess, with a direct line split to either a drain or water faucet. The valve can be capped with a flat acrylic insert to eliminate unnecessary scattering. The insert has an embedded iron piece for easy removal with a magnet. Tap water is used to fill the tank, with the water level replenished using distilled water as gradual evaporation occurs in order to maintain control over the water properties and thus the speed of sound. Distilled water replaces the evaporated water without introducing increased calcium hardness or other changes to water properties. Since distilled water is mineral depleted, the tank is never filled entirely with distilled water which is highly corrosive, especially to metals such as those associated with the body of some underwater transducers and the transducer mounts. A four-stage debubbler and filtration system was developed by John Ellsworth

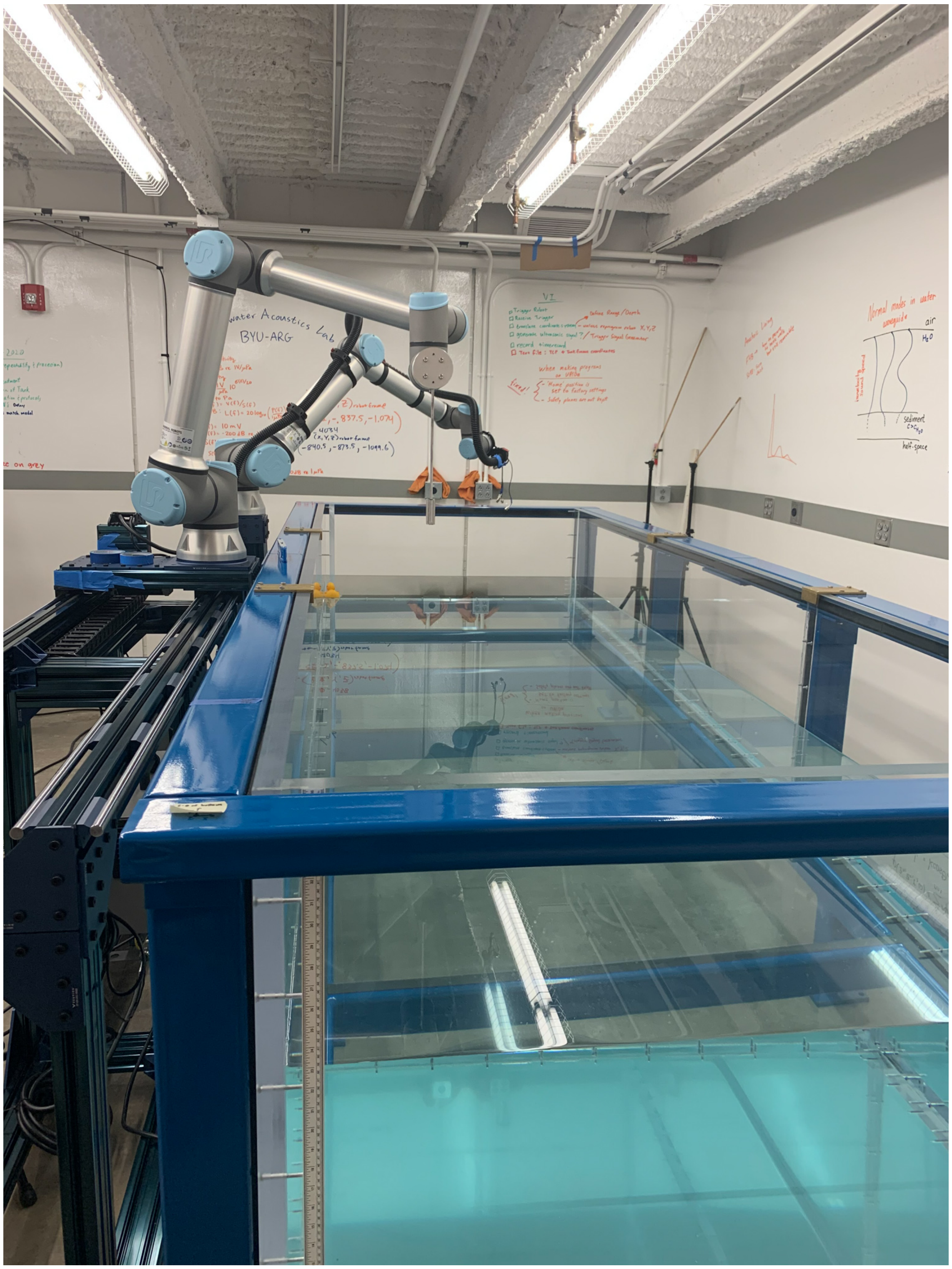


Figure 2.2 View of the BYU acoustic water tank with bare acrylic walls and floor clearly visible alongside the UR10e robot arm positioning system.

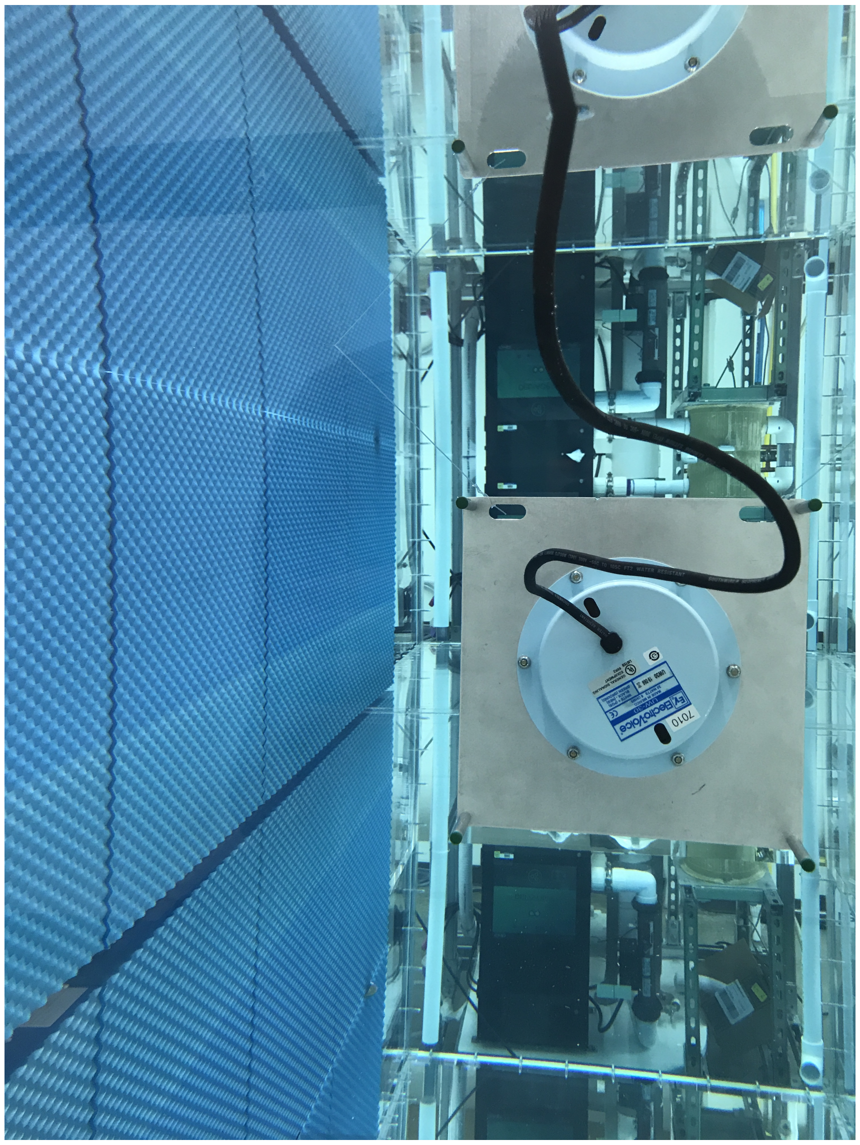


Figure 2.3 View inside BYU's underwater acoustic tank where the left side walls are lined with a blue acoustic attentuating Apltile SF5048 material from Precision Acoustics. This lining is optimized for low-frequency ultrasonic test tanks. The acrylic back wall and floor of the tank can be seen unlined. Also pictured is a pool speaker used for demonstrating principles of underwater acoustics in the audible range.

to maintain the water quality along with chemical treatment when needed. This filtration system was designed to also manage saltwater and work with various added seabed materials. For more details on maintaining the tank, including chemical treatment, see Appendix C.1.1.

2.3 Underwater Positioning System

A simple, safe and reliable 3-dimensional positioning system is achieved with robotic arms and custom software developed in LabVIEW to automate the process.

2.3.1 Robotic Positioning

Two UR10e collaborative robots from Universal-Robots (universal-robots.com) were installed, with one on a Vention (vention.io) 7th axis extender track for increased range of motion. The robots were chosen for their intuitive programming language, high level of programmable safety, and 0.01 mm precision. Each robot operates using six axes of motion and has a maximum physical reach of 1.3 m. Both robots are mounted level with the top of the tank; one on a simple pedestal and the other on the Vention 7th-axis extender track with a rack and pinion motor providing an additional 1.4 m of motion along the length of the tank. The extender track has an added positioning error of ±0.01 mm.

Transducers may be attached to the UR10e in any orientation via custom designed mounts, referred to as tools. Staying within robot tool safety limits, the tools provide an increased reach of 0.5 m and are fitted with emergency float switches to ensure the robots, as seen in Fig. 2.4, are never in any danger of water damage . This feature allows for more flexibility than traditional two or three axis positioning systems while maintaining similar precision [7, 30]. As discussed in Appendix C, smooth transducer motion and orientation control may be performed either via the robot's native Polyscope software or remotely through custom software developed in LabVIEW.

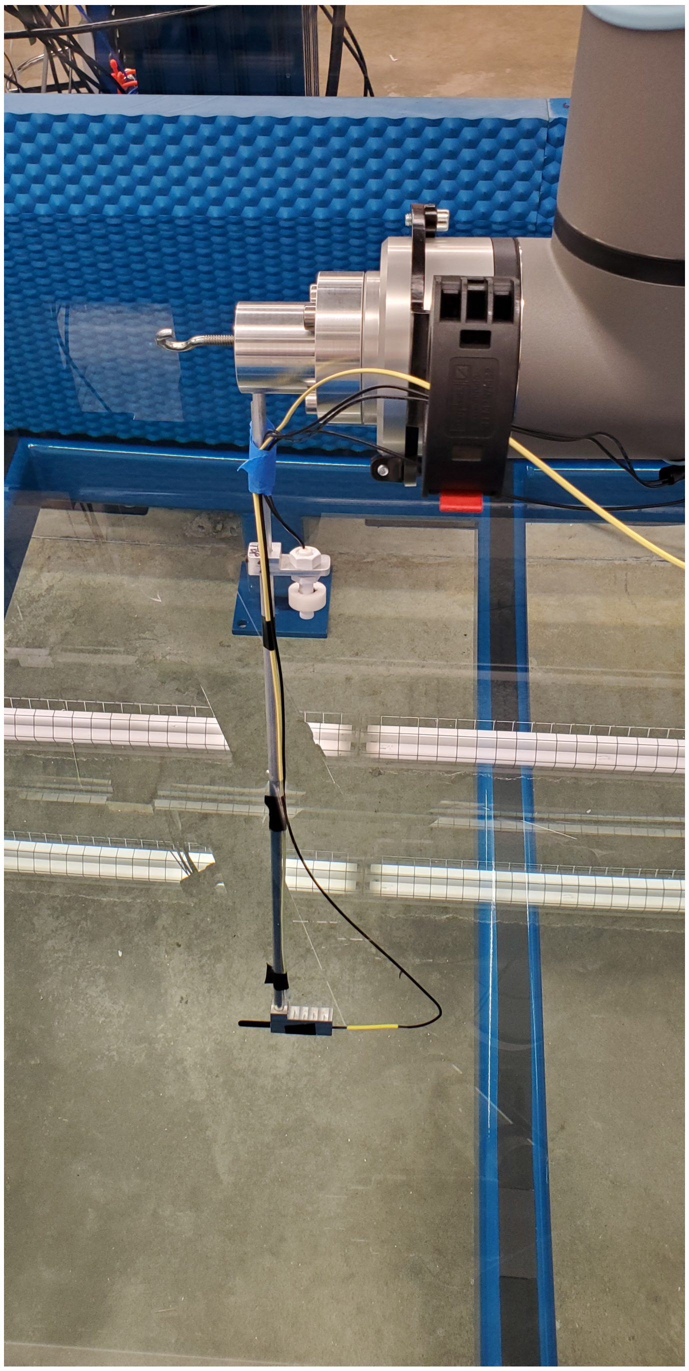


Figure 2.4 UR10e robotic arm with custom tool for mounting and suspending hydrophones within the tank. The white ring toward the top of the rod is a float sensor which when switched will immediately terminate robot movement preventing water damage to the robots.

2.3.2 ESAU

Measurement automation is controlled through TCP/IP by ESAU (Easy Spectrum Acoustics Underwater), a custom software was developed for robot motion control, signal generation and data acquisition. Robot safety limits are also directly hard-coded into ESAU. User-input coordinates in ESAU are sent to the robots and displayed in the 3D plot on ESAU's user interface. An example of the user-interface is shown in Fig. 2.5. Available tool positions/orientations are allowed based on safety limits. ESAU performs an interpolation between the requested positions (Cartesian positions or grid), and any positions outside those limits can be removed in order to scan the largest possible range in the tank. Care has been taken in developing ESAU safety parameters to maintain consistent orientation relative to transducer directivities throughout a motion control sequence. Individual transducers may be selected in ESAU for precise offset positioning of source and receiver relative to the acoustic centers of each hydrophone.

Various signals such as swept-sine waves (linearly or logarithmically frequency modulated sine waves), pure sine waves, pulses, or custom signals may be generated in ESAU for output. Data are acquired from hydrophones through ESAU, which saves the data and displays the time waveform or frequency spectrum of the recorded signal for immediate evaluation. This software was developed with the help of BYU graduate student Adam Kingsley under my direction to meet the needs of the lab.

Further information on the safety provided by and development of the ESAU software may be explored in Appendix C.2.

2.4 Data Acquisition System

Data acquisition and signal generation is performed using hardware from Spectrum Instrumentation and ESAU. The data acquisition cards have a high resolution (16-bit) and are capable of a high

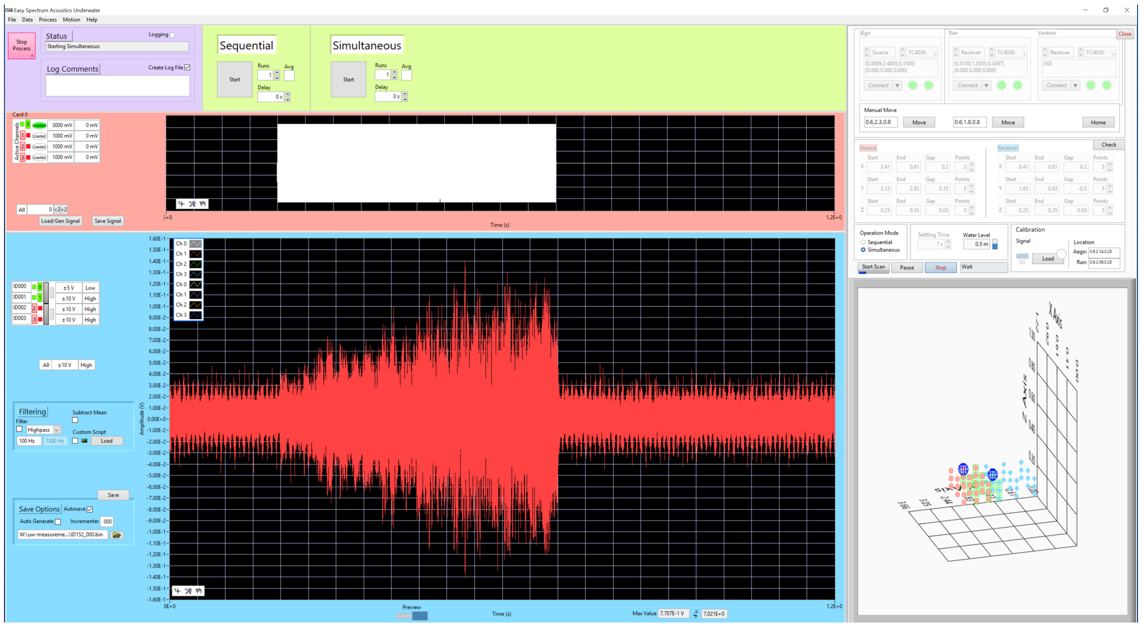


Figure 2.5 User interface of ESAU (Easy Spectrum Acoustics Underwater), which is the custom LabView software developed for signal generation (upper left, red background), data acquisition (bottom left, blue background), and transducer positioning (right panel). In this view, a swept-sine ultrasonic signal is generated (white line/block) as the UR10e robots move the source and receiver positions over a large scan grid (right panel). An example of a received signal is shown as the red line.

sampling rate (40 MS/s). Using the Star-Hub module, the arbitrary waveform generator (AWG) (M2p.6546-x4) and digitizer (M2p.5932-x4) cards are synchronized while housed inside an external PCIe chassis. As implemented, this configuration allots 128 mega samples for each of the four input and four output channels on each of two chassis, which may be daisy chained for use with larger arrays of sources or receivers. The chassis are connected to each other and the control machine via Thunderbolt 3 ports for speed of data transfer. ESAU saves data to the desktop computer as binary files along with associated text files logging the measurement settings for use in later processing code, which are discussed in Appendix B.

2.5 Measurement Chain

2.5.1 Transducers and Amplifiers

The automated measurement system can be used with a variety of transducers. Current measurements are using Brüel & Kjær 8103 phase matched, Teledyne Reson TC4034 and Teledyne Reson TC4038 reciprocal hydrophones for both transmitting and receiving due to their relatively flat response from 4-100 kHz, 5-300 kHz and from 100-500 kHz respectively. Depending on the manufacturer, the receivers are connected to either Teledyne Marine Reson VP2000 EC6081 mk2 [31] or Brüel & Kjær NEXUS type 2692-0S2 conditioning preamplifiers, whose output is sent to the Spectrum cards.

The signal is sent from the AWG through a power amplifier (TEGAM Model 2350) to the transmitting transducer. The TEGAM power amplifier allows a maximum of 4 Vpp input and provides a gain of x50 V. To get a flat response, the TEGAM output may need to be passed through a transformer fabricated to address the frequency-dependent impedance mismatch often found between an amplifier and a piezoelectric source [22]. Each impedance-matching transformer is specifically designed and built in order to provide the flattest response across the widest bandwidth

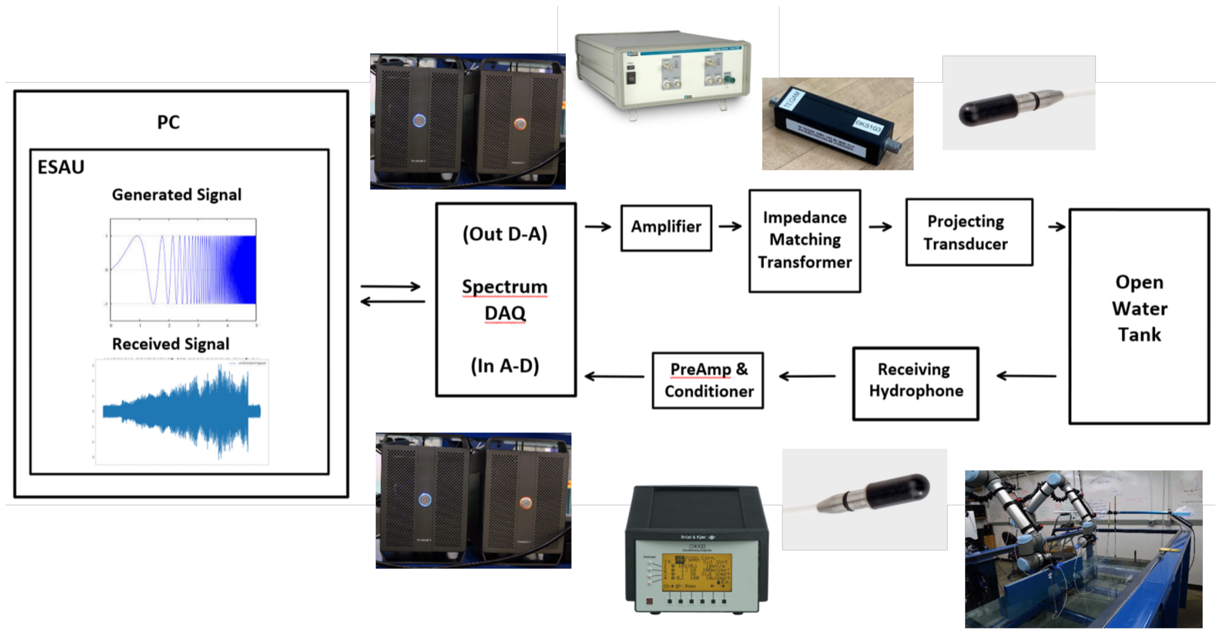


Figure 2.6 Diagram of the measurement chain using the Brüel & Kjær 8103 hydrophones as source and receiver with an impedance-matching transformer and Brüel & Kjær Nexus type 2692-0S2 conditioning preamplifier.

for each specific transducer type. (The impedance matching transformers are not needed for measurements below 10 kHz.)

Each transformer/transducer in the measurement chain has its own frequency response function. The sensitivities of the transmitting and receiving transducers are known from original manufacturer calibrations, but the sensitivity of many other components is unknown. The sensitivities of measurement chain components may be accounted for through the *in situ* calibration measurement discussed in Sec. 3.2.2; this method accounts for the sensitivities of all transducers, amplifiers, preamplifiers and any other component in the measurement chain that alter the signal.

Calibration measurements are made with the source and receiver positioned close together in order to reduce transmission loss through the water, while also accounting for phase effects. A swept-sine, spanning the frequencies of interest, is broadcast and recorded. This calibration measurement is needed to estimate the frequency response of the measurement chain. This through-the-sensor

(TTS) calibration (Sec.3.2.2) incorporates the sensitivities of unknown components [7, 9, 27, 32]. The frequency response of the measurement chain is obtained from the time-gated response of the cross correlation or by phase-corrected deconvolution [27, 33–37].

Each transducer's custom tool mount is made of a thin aluminum rod extended from the robot to maintain orientation of the transducers and maintain the benefit of each transducers' mostly omnidirectional characteristics. The custom transducer mounts allow for multiple configurations such as source or receiver arrays and the addition of a wire thermocouple to measure temperature [11] without significantly increased scattering near source and receiver positions.

General conditions of the tank environment are also monitored with two SensorsOne LMP 307T temperature and pressure/depth sensors from MCT RAM (mctram.com), rated for 0-86 °F and up to 250 m depth positioned at either end of the tank. From these added sensors, underwater environmental characteristics may be effectively monitored. Cables connecting transducers and sensors to the data acquisition system run along the length of the robotic arms, through a cable management system that provides organization and prevents tangling. Consideration has been given to the potential need to shield cables from induced noise coming from either robot motors and brakes or other sources of electromagnetic radiation.

2.5.2 Validation of Measurement Chain

The first step in validating the measurement chain is generating a strong signal in the water. Pure impulses are difficult to generate consistently in underwater acoustics. Preliminary measurements confirmed that long-duration swept-sine signals (chirps or frequency modulated signals) provide the best Signal-To-Noise ratio (SNR) for broadband measurements [7, 27, 35, 37] compared with white noise, pulses [10], or averaging short swept-sine signals.

To understand the transfer function and, therefore, sound propagation in the water tank, recorded swept-sine signals over the bandwidth of interest have been processed via cross correlation or frequency deconvolution to obtain the impulse response of the system [33, 37–42]. This method of frequency deconvolution is discussed further in Sec. 3.2.1. Long-duration sine waves have also been used effectively to validate the measurement protocol within the tank at specific frequencies. Both long-duration swept-sine signals and sine waves allow for significant energy to enter into the tank and provide good SNR [7, 27, 35, 37] since the energy can reach steady state.

The next step in the validation process was to evaluate the effectiveness of the choice to use acrylic walls and the impact of the Apltile SF5048 attenuating panels. Scans were performed with and without panels, and comparisons between measurements quantify the reduction in signal reflections and corresponding reduction in reverberation time [10, 26, 37]. Initial results from these swept-sine measurements, performed across the tank, provided an initial validation. For positions near the walls, the impulse response of the 100-500 kHz swept-sine signal showed a reduction of about 8 dB comparing single-bounce reflections off the sidewalls with and without the attenuating panels. Also observed was about a 3 dB reduction on the overall sound pressure level (OASPL) of the time-gated first reflection. This reduction in sound energy from reflections helped to quantify unnecessary delay between measurements. More accurate and full evaluation of the reverberant characterization of the water tank with and without attenuating materials is discussed in Ch. 4; the methods to determine these characteristics are discussed in Sec. 3.1.2.

2.6 Limitations and Improvements

An open-air water tank of this size has numerous potential applications, however, limitations do exist. Because of lateral reflection, in most cases, the tank is effectively limited to time-gated, scaled measurements. The side walls, maximum depth and robot arm reach limit the range of source-receiver positions for potential scaled experiments [6, 7, 30]. Time-gating may be performed to improve upon limits due to the dimensions of the tank. Time-gating is done using arrival times determined through time of fight by the method of images. However, in applications attempting to model scaled open-ocean measurements, time-gating can still be limited when source or receiver positions are too close to side walls and if early reflections from the water-air surface or "seabed" arrive after unwanted early side wall reflections.

Potential regimes for scaled experiments are limited by the frequency response of the transducers and dimensions of the tank in accordance with the desired scaling factor. The size of transducers, mounts, etc. must also be considered when designing the experiment as any objects that are large relative to the wavelength of interest become potential scatterers.

The selected anechoic or attenuating material is not perfectly anechoic at all frequencies as would be expected, and therefore the tank cannot perfectly model a scaled, open-ocean environment. However, the addition of attenuating panels along the side walls does increase efficiency by reducing the needed delay between consecutive measurements. In addition, the available attenuation or absorption may improve models for open-ocean measurements in tanks as discussed in Sec. 3.3, 3.1.4 and Ch. 5. Passive underwater attenuating lining is often optimized for ultrasonic measurements and can be very costly, though other attenuating treatment options are available [5–7, 10, 21, 31] for a variety of bandwidths. Multi-layered treatments, wider tanks, and active acoustic absorbers could improve the anechoic nature of the tank but also increase overall costs.

The positioning system utilizes UR10e robots to effectively scan the majority of the tank with any transducer orientation. Scan positions are limited less by the effective reach of the robots than by the need to time-gate side reflections while maintaining desirable surface and floor reflections. A tank with larger dimensions would allow for better measurements at lower frequencies and increase potential applications but also require more creative solution for the positioning system [22, 27] than what has been chosen here. This increased size would also require a significantly larger lab space, which would be even more difficult to come by. The current positioning system could be improved by having both robots on 7th axis extender tracks potentially on a gantry system above the tank to provide greater flexibility to reach more locations within the dimensions of the tank.

2.7 Summary

An underwater acoustic laboratory measurement system has been described with the goals of capturing large datasets efficiently in a controlled environment while helping students develop effective experimental research skills. High priority has been given to developing a measurement system that is capable, reliable, easy to use, and safe. Considerations that guided the design included the dimensions and materials of the tank, the capabilities of the data acquisition system, the precision and automation of the positioning system, and effectiveness of the measurement chain. Potential limitations have been discussed.

The dimensions of the tank are the primary limiting factor in what underwater acoustical measurements can be taken effectively in the laboratory. The width of the tank determines the effectiveness of time gating to leave only the direct path and first surface and bottom reflections. The attenuating materials reduce reverberation time and help to reduce some limitations to the dimensions of the tank. Though there are some limitations associated with the use of LabVIEW controlled robotic arms as the positioning system, they offer an alternative solution to traditional linear motion Cartesian positioning systems while allowing active control of transducer orientation with respect to directivities and maintaining high precision motion. The custom LabVIEW software, ESAU, was specifically developed with the idea of meeting the above concerns while maintaining versatility in generating signals, controlling motion, and recording measurements. Many transducers are available for tank measurements but require further consideration in order to prevent adding acoustic scattering sources to the tank due to the transducers physical size relative to the wavelength. Each transducer type is also optimized for specific bandwidths. Impedance matching transformers have been custom built in order to improve on some of these limitations by providing a flatter response with regard to the rest of the measurement chain. These concerns determine the effective bandwidths and need for further considerations. The data acquisition system was designed to meet the demands of the ultrasonic bandwidths that perform best in the laboratory environment. These systems can handle the high processing demands of this research. An understanding of the limitations and capabilities of the measurement chain and tank environment have opened the way for future measurement applications and a more full evaluation of the nature of acoustical measurements in a laboratory-tank environment.

Chapter 3

Methods

3.1 Tank Measurement Considerations

The realities, considerations, limits and methodology of taking acoustical measurements in a water tank are discussed in this section. This section includes physical limitations of acoustical measurements in an enclosed tank environment, how time-gating is performed, and the acoustic characterization of the tank, especially related to boundary conditions. With proper acoustic characterization of the water tank, effective models for sound propagation may be developed and acoustic tank measurements may be related to a scaled-ocean model.

3.1.1 Special Considerations and Limits

As mentioned, water tanks can be used for validating measurement techniques and propagation models. One upcoming research plan includes gathering large data sets for the testing of machine learning. To consider the tank as a scale-model of the ocean, with typical operating bandwidths of 100-200 Hz, frequencies are scaled by a factor of 100-1000 (10-200 kHz, for example). However, sound propagation in the tank will not be a perfect scale-model of the ocean for several reasons, particularly when acoustic properties such as transmission loss are not scalable quantities.

3.1.2 Characterization

The primary factor limiting scaled-ocean measurements are the dimensions of the tank. The tank is a rectangular parallelepiped with inner dimensions of 1.22 m wide by 3.66 m long by a maximum of 0.92 m deep. The geometry of the tank, the reflections from the acrylic walls, and the water depth dictate some properties of acoustic propagation including the Schroeder frequency, reverberation time, and absorption.

The dimensions of the tank and the reverberation time determine the Schroeder frequency, which dictates the cutoff frequency between the low-frequency region and the mid to high-frequency region, which acoustically behave very differently. The low-frequency regime is where the acoustic energy is dominated by discrete resonances. In the mid to high-frequency regime, the acoustic energy becomes more diffuse. The Schroeder cutoff frequency for the tank was estimated as about 1000-3000 Hz dependent on the water depth, sound speed, and actual acoustic absorption of the tank walls. This cutoff frequency is important because the acoustical measurements for a scale-model for open-ocean sound propagation are those above the cutoff frequency, i.e., the mid-to high-frequencies. The estimated Schroeder frequency is determined by Eq. 3.1 [43], where c is the speed of sound in water, T_{60} is the estimated 60 dB reverberation time, and V is the total volume of the water tank:

$$f_{\text{Schroeder}} = \sqrt{\frac{c^3 T_{60}}{4 \ln 10 V}}. \tag{3.1}$$

To obtain the estimated Schroeder frequency, the reverberation time or T_{60} must first be determined by the Norris-Eyring equation [44] modified to account for propagation absorption:

$$T_{60}(f) = \frac{(24 \ln 10)\, V}{c\{-S \ln[1 - \langle \alpha(f) \rangle_S] + 8\, \alpha_p(f)\, V\}}, \tag{3.2}$$

where S is total surface area, $\langle\alpha(f)\rangle_S$ is the frequency-dependent spatially averaged absorption coefficient of the boundaries, and $\alpha_p(f)$ is the frequency-dependent absorption coefficient from acoustic propagation through the water. The modified Norris-Eyring equation is used instead of the Sabine-Franklin-Jaeger model, which does not perform as well for enclosures with large wall absorption [44] as anticipated within the tank environment.

The spatially averaged absorption coefficient is typically determined experimentally as discussed in Sec. 3.1.4, and the propagation absorption can be estimated using one of many underwater acoustic propagation absorption models. Ainslie and McColm [45] simplified Francois and Garrison's [46, 47] formulation, which was developed after Fisher and Simmons [48], for acoustic propagation absorption through water with respect to frequency (f) in Hz, water acidity (pH), water temperature (T) in °C, water salinity (Sal) in parts per thousand (ppt), and depth (z) in km:

$$\begin{aligned}\alpha_p = 0.106\,&\frac{f_1 f^2}{f^2+f_1^2}\,e^{(\mathrm{pH}-8)/0.56}\\ +0.52\,&\left(1+\frac{T}{43}\right)\left(\frac{\mathrm{Sal}}{35}\right)\frac{f_2 f^2}{f^2+f_2^2}\,e^{-z/6}\\ +0.00049\,&f^2\,e^{-(T/27+z/17)}\end{aligned} \tag{3.3}$$

where,

$$f_1 = 0.78\,(\mathrm{Sal}/35)^{1/2}\mathrm{e}^{\mathrm{T}/26} \tag{3.4}$$

$$f_2 = 42\,e^{T/17} \tag{3.5}$$

Propagation losses are important for more precise modeling and are represented in normal-mode models as the imaginary part of the modal eigenvalues. For this work, Ainslie and McColm's propagation absorption model is evaluated, in addition to a model that assumes zero absorption, because of the simplicity of the model over others.

Another slight limitation of using the Norris-Eyring equation (Eq. 3.2) was explained by Müller-Trapet [36] who showed that a recording cannot simply be stopped at the end of a swept-sine signal. In order to obtain the reverberation time, a recording must allow for a certain amount of trailing zeros within the signal or a "stop margin" to capture sufficient high-frequency energy decay at the end of the swept-sine signal. This "stop margin" can be calculated from the reverberation time of the highest frequency within the swept-sine signal and ensures that ample recording time is performed to capture the full signal appropriately. To accomplish this goal, the generated signal should contain zeroes over this "stop margin" after the swept-sine wave signal.

Estimations of T_{60} and the Schroeder frequency are key to providing a starting point or benchmark in characterizing the tank environment. A benchmark T_{60} estimation is necessary to address the semi-subjective nature of applying reverse Schroeder integration [41, 42, 49, 50] to obtain the decay curve as discussed in Sec. 3.1.4. To obtain more precise quantities from measurements, effective time-gating techniques are required as well as methods for obtaining an impulse response from swept-sine signals, as explained in the following sections.

3.1.3 Time-Gating Signals in the Tank Environment

The first step in obtaining measured T_{60} quantities from the impulse response of the tank environment by the process discussed in Sec. 3.1.4 is to obtain a calibrated response of the measurement chain, which can be used to account for all frequency-dependent sensitivities and electronic effects on the signal. The calibration measurement requires the source and receiver to be close so as to minimize the impact of the tank environment. The calibration signal is time-gated to remove reflections from the walls and bottom of the tank and the water surface, so as to include only effects of the measurement chain on the response of the signal. This system response can then be accounted for in all other recorded measurements. The motivation and methodology for this calibration are detailed in Sec. 3.2.2. Care must be taken to time-gate the recorded signals appropriately since

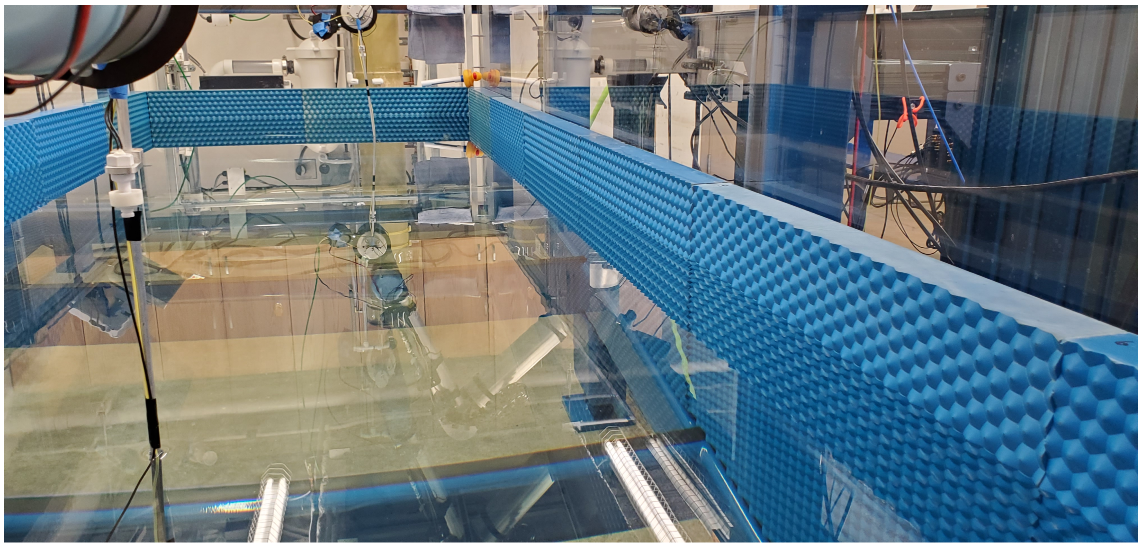


Figure 3.1 BYU's underwater acoustic tank walls lined with blue acoustic attentuating Apltile SF5048 material from Precision Acoustics on the side walls. This lining is optimized for low-frequency ultrasonic test tanks.

acoustic reflections from the walls of the tank cannot be perfectly mitigated to match an open-ocean environment even when reflections are reduced by a 5 cm thick anechoic (acoustic attenuating) material placed on all the walls of the tank as seen in Fig. 3.1.

Time-gating is simply the process of truncating the signal before unwanted reflections arrive. The time it takes for reflections to arrive may be calculated using the method of images and ray tracing, as illustrated in Fig. 3.2. Predicting time of flight by the method of ray tracing is particularly accurate for frequencies above the Schroeder cutoff frequency where acoustic energy behaves more like rays and predicting arrival times of acoustic signals is most effective.

When time gating, a half-Hanning window should be applied to the recorded signal in order to truncate the signal after the the arrival of the direct signal at t_d but before the arrival of the first wall reflection at $t_{1\mathrm{r}}$. Application of a half-Hanning window prevents discontinuities within the time-gated signal, which would cause high-frequency noise artifacts to appear as a result of immediately truncating the signal. However, since actual measured signals may also contain a pre-ringing artifact, the half-Hanning window used for time-gating the signal must allow for a

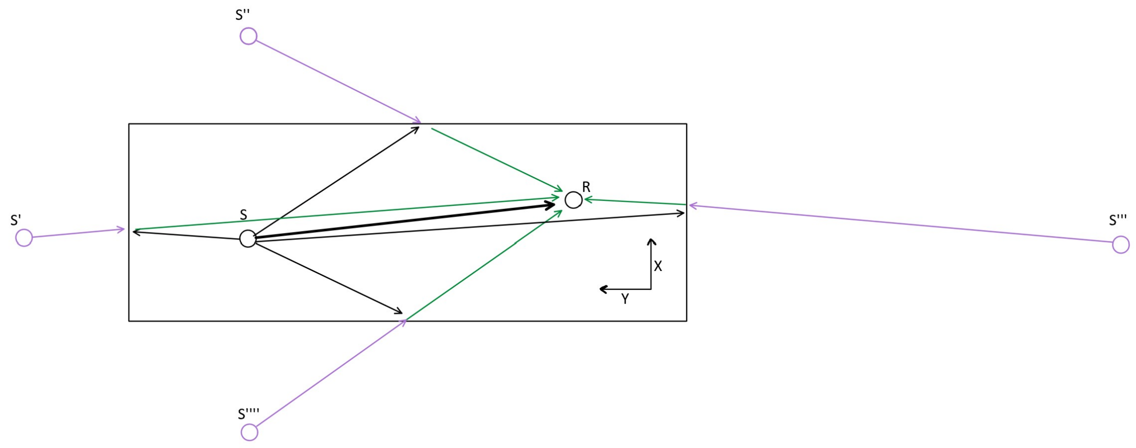


Figure 3.2 Method of images used for predicting reflections via ray tracing. R represents the receiving hydrophone position, S represents the source position and S', S'', S''' and S'''' represent image sources from single wall reflections.

buffering time (δt) adjusting when the windowing occurs such that the half-Hanning window is applied at time $t_{1\mathrm{r}} - \delta t$. This ensures that the windowing truncates the signal before pre-ringing artifacts occur from the first wall reflection as seen in Fig. 3.3.

Time-gating the signal in the tank may be imperfect when used on non-impulsive signals. However, a swept-sine wave or chirped signal may be time-gated after obtaining the impulse response of the signal through frequency deconvolution [27,33–37], as discussed in Sec. 3.2.1 using the codes contained in Appendix B.2.1.

3.1.4 Boundary Characterization

An important tank characteristic that is needed to model sound propagation in an environment with finite-impedance boundary conditions is the frequency-dependent, spatially averaged boundary absorption: $\langle\alpha(f)\rangle_S$. Unlike Li *et al.* [29], who measured the impedance boundary of their tank walls via plane-wave tube measurements, the work in this thesis calculates the approximate frequency-dependent absorption coefficient of the boundaries from the measured T_{60} by solving the modified Norris-Eyring equation (Eq. 3.2) with no propagation absorption as well as evaluating the benefit

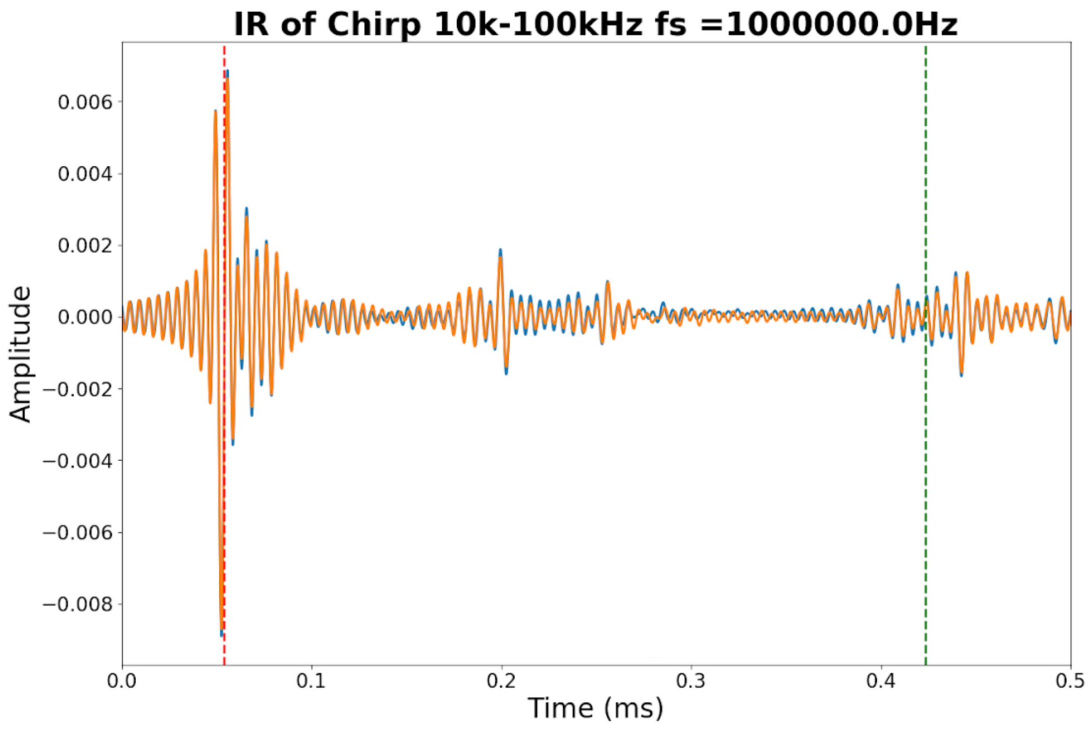


Figure 3.3 Two recorded swept-sine signals (orange and blue line) with predicted direct arrival t_d (red dashed line) and predicted first reflection $t_{1\mathrm{r}}$ (green dashed line). This figure demonstrates the importance of offering a buffer time, δt before the time gating to truncate before any pre-ringing artifacts of the signal seen before the green dashed line.

of accounting for propagation absorption. When boundary treatment is added, such as the use of the anechoic panels discussed in Sec. 3.3.5, the comparison between measured T_{60} values can quantify the corresponding change in boundary absorption [10,23,26,37] and characterize boundary materials. The technique used in ISO 354 [50] to characterize boundary absorption and often referred to as the reverberation method for Sabine characterization used in aerial room acoustics has been used in many test tanks [10,23–26], though not extensively. This thesis provides an evaluation of this method over much more than 12 source and receiver positions as is required according to ISO 354 and further offers an initial evaluation of the application of propagation absorption, as mentioned above, which is not commonly applied in previous studies.

The estimation of $\langle\alpha(f)\rangle_S$ relies on an estimation of the T_{60}. The T_{60} is obtained from the impulse response of the tank, as discussed in Sec. 3.2. The impulse response squared, $h^2(t)$, is then passed through a fractional octave filter [51]. Finally, Reverse Schroeder integration [49], following the integrated impulse response method of ISO 3382-1:2009(E) [41], is performed to determine the T_{60}. Details of the calculations are provided in B.3.5 .

With the frequency dependent T_{60}, a spatially averaged, frequency-dependent absorption coefficient, $\langle\alpha(f)\rangle_S$, is calculated by solving Eq. 3.2. The obtained value of $\langle\alpha(f)\rangle_S$ can be applied as the finite-impedance boundary condition necessary for an effective acoustic propagation tank model, as discussed in Sec. 3.3.

3.2 Impulse Responses Measurements in a Water Tank

Impulse response measurements in small water tanks bring a multitude of challenges to be considered, especially when striving to develop a scale model of the open ocean. Because this goal requires special consideration of wall reflections, consideration must be given to the type of signal used to excite the medium for measurement of the impulse response.

Ideally, a Dirac delta function would be used as the source signal for impulse response measurements. The Dirac delta function, which is infinitely narrow in time and large in amplitude, contains energy at all frequencies; however, a perfect Dirac delta function is impossible to generate physically. Because of this, short pulsed signals or other short impulses, such as those generated by balloon explosions, starter pistols, electric discharge, and breaking glass, have often been used in room or underwater acoustics. [21] Another approach utilizes the interrupted noise method to obtain a broadband signal according to ISO3382-1:2009 [41].

ISO3382 [41,42] and ISO354:2003 [50] state that using a deterministic signal such as a swept sine wave to obtain an impulse response ($h(t)$) in room acoustics yields improved signal-to-noise ratio (SNR) [52] after special processing of the recorded signal. With improved computational abilities, impulse response (IR) measurements using swept sine waves in water tanks are becoming increasingly commonplace. [28,34] A swept sine wave consists of a sine wave with a frequency that linearly or logarithmically increases with time, as seen in Fig. 3.4.

Several researchers have confirmed that long-duration swept-sine signals provide better SNR for broadband measurements [7,27,35,37] compared with white noise, impulses, and pulses. [10] This high SNR is a result of generating more energy equally over all frequencies, where the time duration determines the SNR. [35] This high SNR from long-duration swept-sine signals is accomplished even without averaging in accordance with ISO18233/2006 [52]. The use of long-duration swept-sine signals avoids the problems that can be caused when using multiple averages with time invariant systems. [40] Swept-sine signals are repeatable as deterministic signals unlike background or measurement noise, or other common signals mentioned above which are not time-invariant.

Since swept-sine signals provide high SNR and are repeatable, they should be an effective signal for obtaining an accurate broadband response of a water tank just as they do for room environments, assuming post processing of the recorded signal is done according to ISO 18233:2006 [52]. Accurate

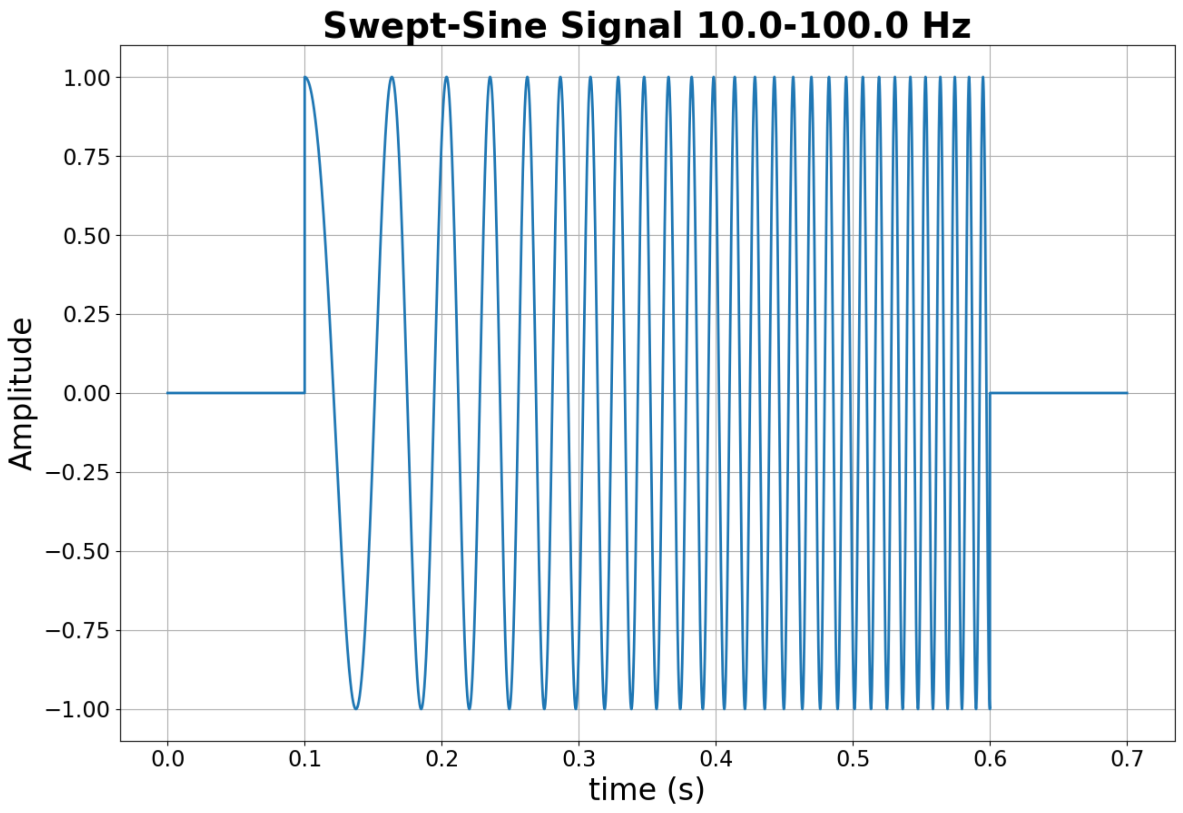


Figure 3.4 A 0.5s swept-sine wave signal (chirp) containing frequencies 10-100Hz.

impulse response measurements allow for determining many important acoustical characteristics of the water tank, as well as providing an effective comparison to and evaluation of numerical models of sound propagation in the water tank, which are needed for future research.

3.2.1 Theory

In order to process a swept-sine signal effectively, the nature of a recorded swept-sine signal in an enclosure must be understood. An input signal $g(t)$ in an enclosure interacts with the enclosure environment to produce the received signal $r(t)$. This interaction with the environment is represented mathematically as a convolution with the impulse response $h(t)$ of the enclosure:

$$r(t) = g(t) * h(t). \tag{3.6}$$

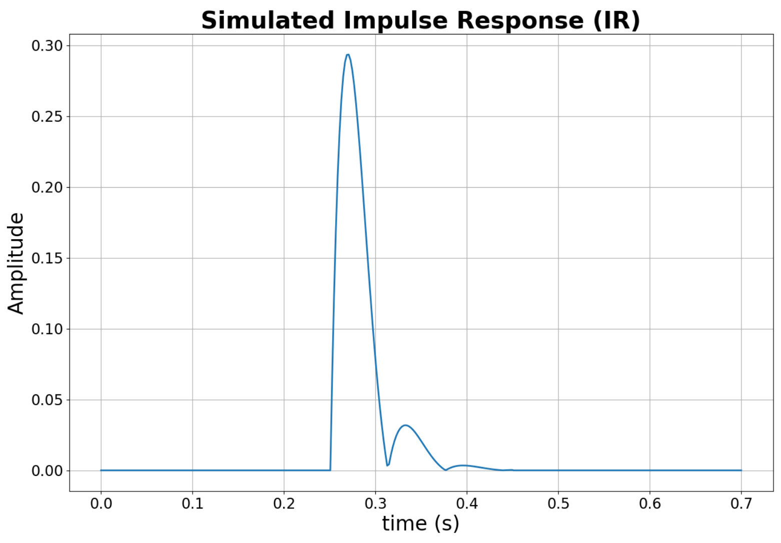
This mathematical principle is illustrated in Fig. 3.5. A swept-sine signal $g(t)$ (Fig. 3.4) is convolved with the impulse response $h(t)$ of a simulated environment (Fig. 3.5a) to produce the convolved simulated received signal $r(t)$ (Fig. 3.5b).

In practice, the impulse response $h(t)$ of the enclosed field is unknown and must be determined. The classical approach to determining an unknown impulse response is accomplished with three steps: (1) Fourier transforms on the recorded signal $r(t)$ and the generated signal $g(t)$ to obtain $R(f)$ and $G(f)$, respectively; (2) frequency deconvolution [27,33–37] to obtain the frequency response function $H(f)$ (FRF), and (3) an inverse Fourier transform to obtain $h(t)$. [36,38,39] Mathematically this approach proceeds as follows:

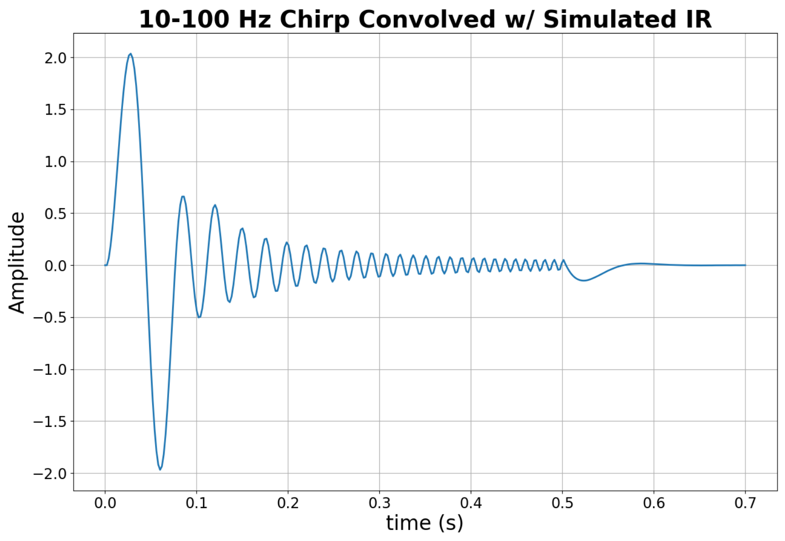
$$F\{r(t)\} = F\{h(t)\}F\{g(t)\} \tag{3.7}$$

$$R(f) = H(f)G(f) \tag{3.8}$$

$$\frac{R(f)}{G(f)} = H(f) \tag{3.9}$$



(a) Simulated IR



(b) Simulated Recording

Figure 3.5 Example of a convolved simulated received signal as a result of the convolution of the swept sine signal Fig. 3.4 with the simulated impulse response in Fig. 3.5a. The resultant simulated recorded was shifted in time in order to see the full convolution.

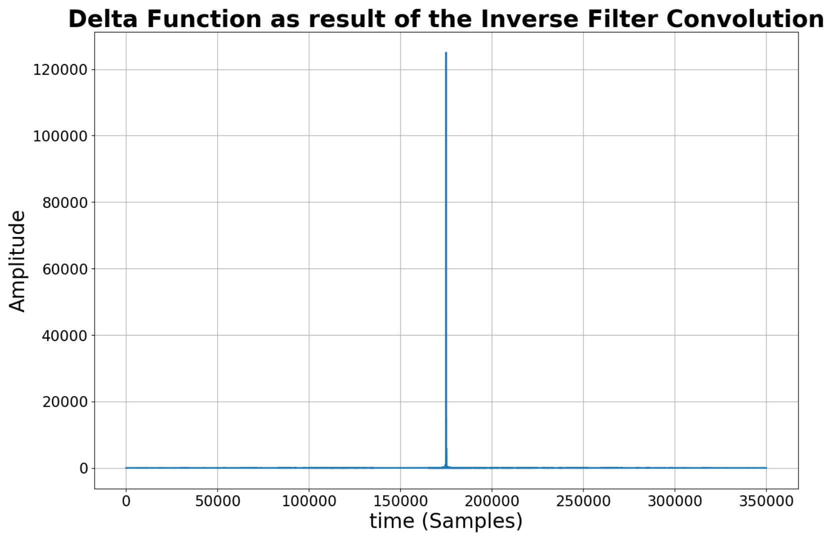


Figure 3.6 Delta function as a result of the temporal inverse filter $g^*(-t) * g(t)$

$$h(t) = F^{-1}\{H(f)\} \tag{3.10}$$

where $F\{r(t)\}$ represents the fast Fourier transform (FFT) of $r(t)$ and $F^{-1}\{H(f)\}$ is the inverse fast Fourier transform (IFFT) of $H(f)$.

This classical approach is commonly used in analytical work. In experimental work, however, a phase inversion from cross-correlation with the temporal inversion of the generated signal can also be applied in order to obtain a pure delay of the impulse response with no amplitude or phase contributions due to the excitation method thus avoiding noise more effectively: [9, 34, 35, 40]

$$g^*(-t) * r(t) = h(t) * g^*(-t) * g(t). \tag{3.11}$$

This approach works because the inverse filter $g^*(-t) * g(t)$ forms the delta function [34], as seen in Fig. 3.6 and applied in Eqs. 3.11 and 3.12 ($g^*(-t)$ represents the conjugate of the temporal inversion of $g(t)$).

After solving for $H(f)$, the inverse Fourier transform yields $h(t)$:

$$F^{-1}\left[\frac{F[g^*(-t)*r(t)]}{F[g^*(-t)*g(t)]}\right] = h(t) \tag{3.12}$$

In practice, this computation is more easily accomplished numerically in the frequency domain; this deconvolution is viewed as the ratio between the cross-spectral density ($S_{gr}(f)$) with the auto-spectral density ($S_{gg}(f)$):

$$H(f) = \frac{G^*(f)R(f)}{G^*(f)G(f)} = \frac{S_{gr}(f)}{S_{gg}(f)}, \tag{3.13}$$

in which the g and r subscripts denote the generated and received signals, respectively. To avoid division by zero, Wiener deconvolution may be used to apply a regularization function:

$$H(f) = \frac{S_{gg}^* S_{gr}}{S_{gg}^2 + \sigma^2}\bigg]. \tag{3.14}$$

The regularization parameter σ is proportional to the mean of the magnitude of S_{gg}:

$$\sigma = \lambda\overline{|S_{gg}|}, \tag{3.15}$$

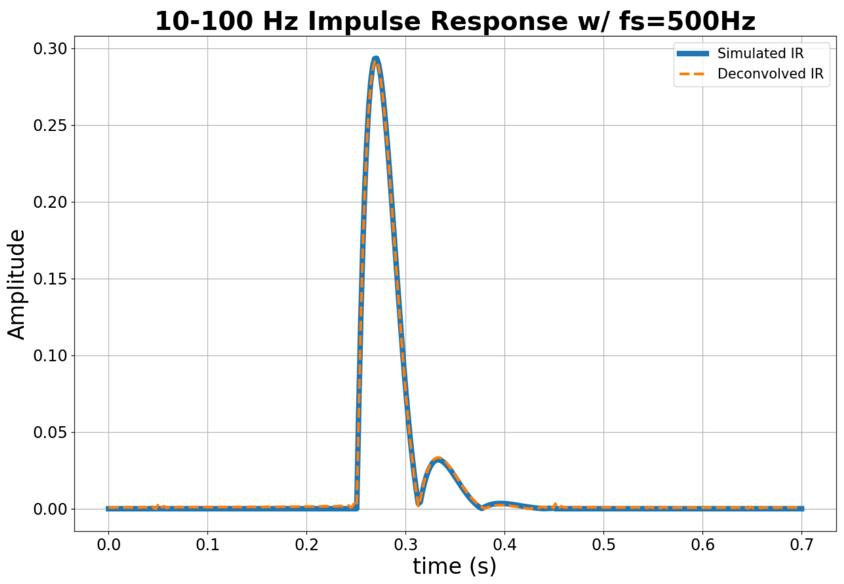
The scaling parameter λ is chosen to keep $H(f)$ from growing rapidly when the auto-spectral density S_{gg} is near zero.

An example of this process is now provided. A swept-sine signal $r(t)$ was generated for 10-100 Hz, 1-10 kHz, 10-100 kHz, and 100-500 kHz bandwidths, with and without noise using various sampling rates. A simulated impulse response $h(t)$ was generated by a Python function from the scipy.signal library. The deconvolution method given in Eqs. 3.12, 3.13, 3.14 was used. The frequency deconvolution method acting on the simulated data, $r(t)$ in Fig. 3.5 yields the impulse response $h(t)$, and frequency response $H(f)$. The results for the 10-100 Hz band are displayed in Fig. 3.7 as the orange dashed lines. For comparison, the impulse response and frequency response used to create the simulated signal are also shown as blue solid lines. The impulse response matches well, as does the frequency response over the 10-100 Hz band. However, above 125 Hz, the frequency response contains a ringing artifact. Additional examples are now provided to demonstrate the capabilities of the deconvolution method at handling various bandwidths and sampling rates, as well as induced noise.

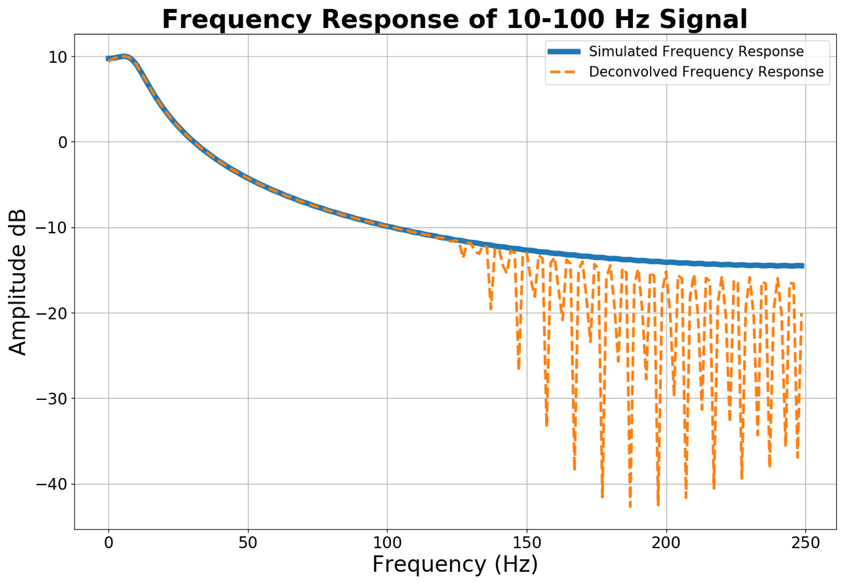
The performance of this deconvolution method on signals in different frequency bands is evaluated using these simulated signals. The resulting frequency and impulse responses are shown in Figs. 3.8 and 3.9, respectively, for the 1-10 kHz band (a), 10-100 kHz band (b), and 100-500 kHz band (c). The frequency responses obtained via deconvolution are very accurate over the band of the generated signal. However, impulse responses for lower bands provide better agreement with actual values with better precision of amplitude and general shape. The agreement of the impulse response is not as good as the 10-100 Hz example in Fig. 3.7, which points to the question of how the sampling frequency impacts this process.

For each of the simulations in Figs. 3.7, 3.8, and 3.9, the frequency bands are well under the Nyquist limit to provide a more well-defined signal, particularly in the time domain. The impact of sampling frequency on the 10-100 kHz band can be seen clearly in Fig. 3.10. The different rows correspond to the impulse response ($h(t)$) and frequency response ($H(f)$) when different sampling frequencies are used in generating the simulated signal. As expected, with increased sampling rate typically comes increased precision. However, sampling too high decreases the precision of the solution. This effect can be seen in the cases of the 10-100 Hz swept-sine signal sampled at 500 Hz and at 500 kHz in Fig. 3.11: Excessively high sampling rate makes the resulting impulse response less precise. This observation suggests there may be a limit to the degree of precision that may be obtained. This limit is not of concern for the measurements used in this thesis.

As mentioned, this deconvolution method also handles noise quite effectively without the need for averaging. An example of this behavior is provided in Fig. 3.12. In this case, a noisy swept sine signal (a) is convolved with a simulated impulse response to produce the received signal (b). This received signal is sent to through the deconvolution process to estimate the impulse response and



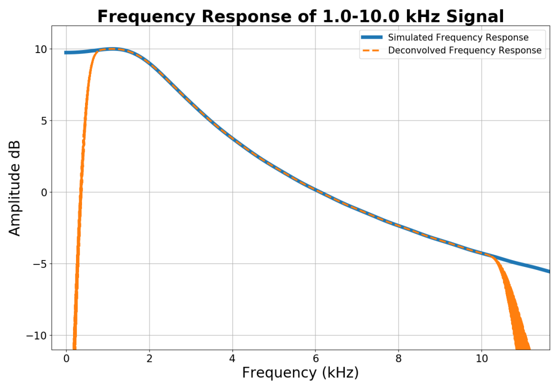
(a)



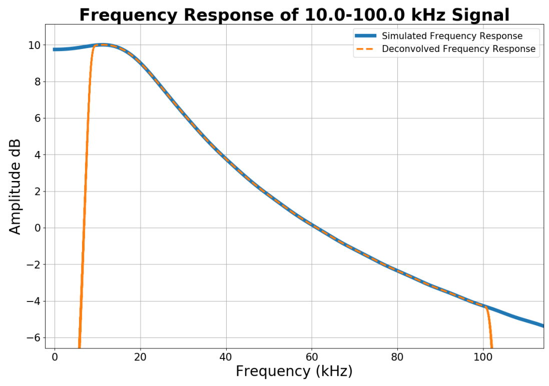
(b)

Figure 3.7 Impulse response (a) and frequency response (b) for a 10-100 Hz swept-since wave in a simulated environment obtained using the Wiener deconvolution method described in Eqs. 3.12, 3.13, and 3.14

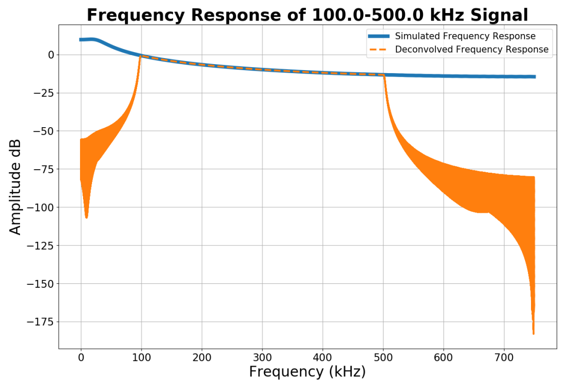
.



(a) 1k-10kHz FRF

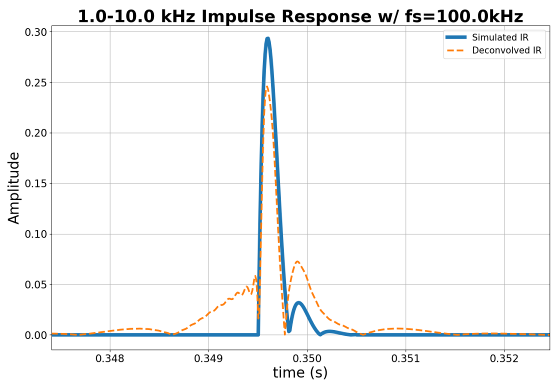


(b) 10k-100kHz FRF

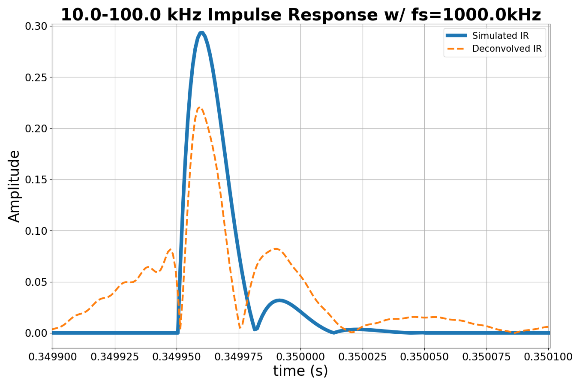


(c) 100k-500kHz FRF

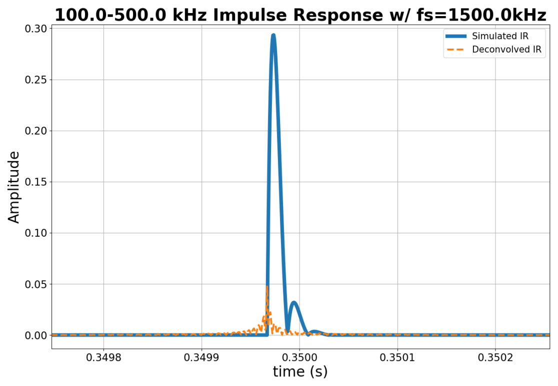
Figure 3.8 Deconvolved frequency responses for simulated swept-sine signal for various bandwidths (orange) compared to the frequency response (blue) used to simulate the signal.



(a) 1k-10kHz IR

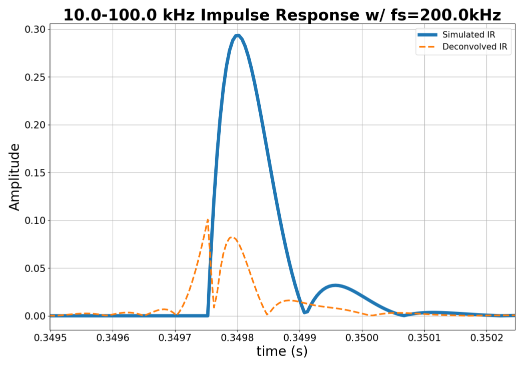


(b) 10k-100kHz IR

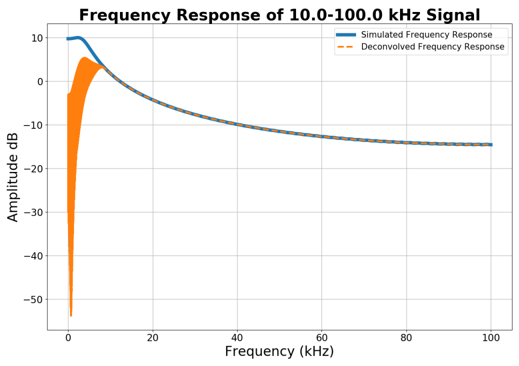


(c) 100k-500kHz IR

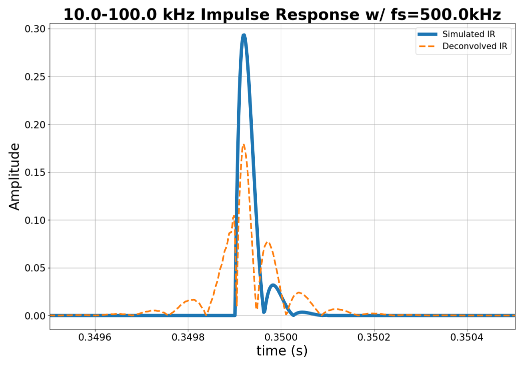
Figure 3.9 Deconvolved impulse responses (orange) of swept-sine signals for various frequency bands compared to the impulse response used to simulate the signal (blue).



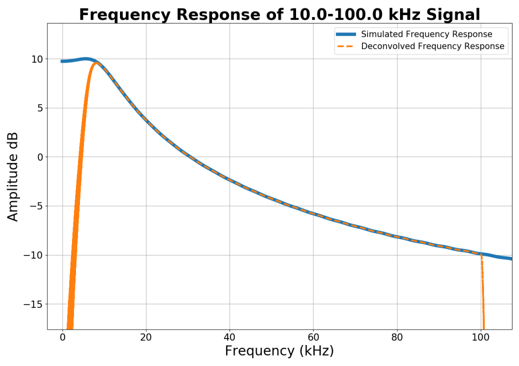
(a) IR w/ fs=200kHz



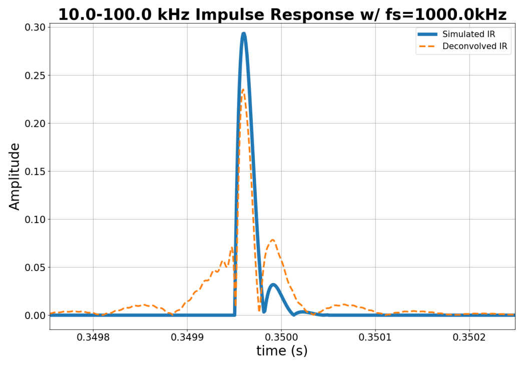
(b) FRF w/ fs=200kHz



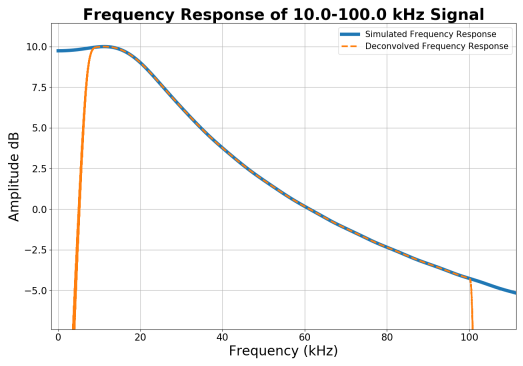
(c) IR w/ fs=500kHz



(d) FRF w/ fs=500kHz

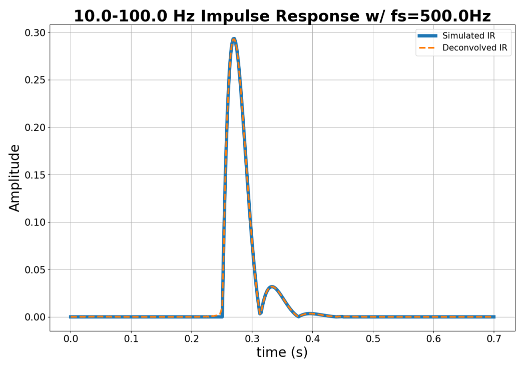


(e) IR w/ fs=1MHz

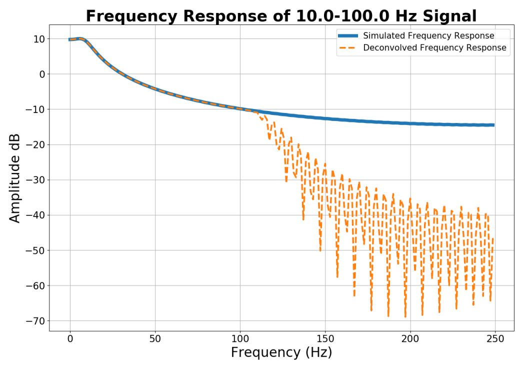


(f) FRF w/ fs=1Mhz

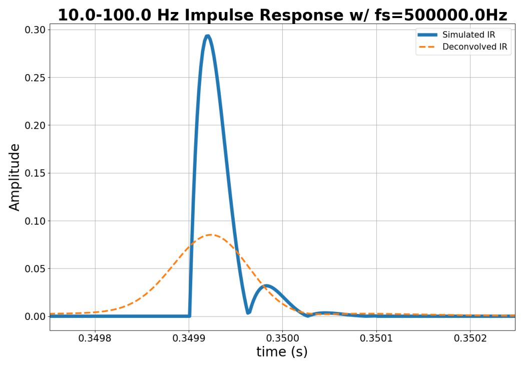
Figure 3.10 Deconvolved impulse and frequency responses (IR and FRF) for a simulated 10-100 kHz swept-sine signal with various sampling rates (f_s).



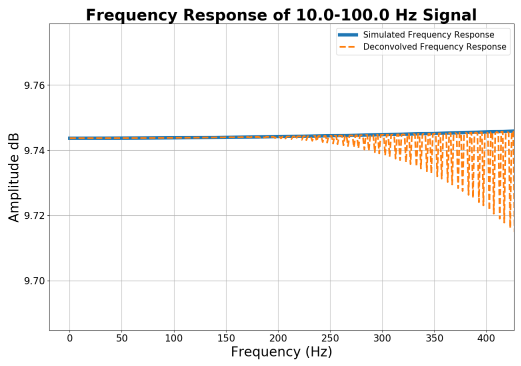
(a) IR w/ fs=500Hz



(b) FRF w/ f_s=500Hz

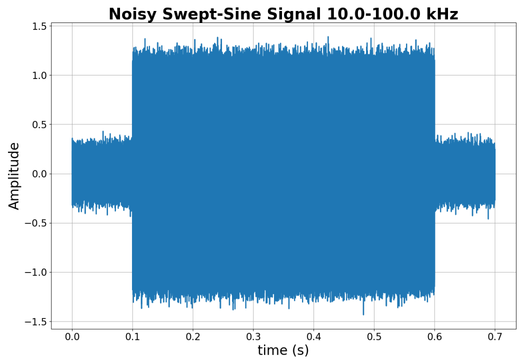


(c) IR w/ fs=500kHz

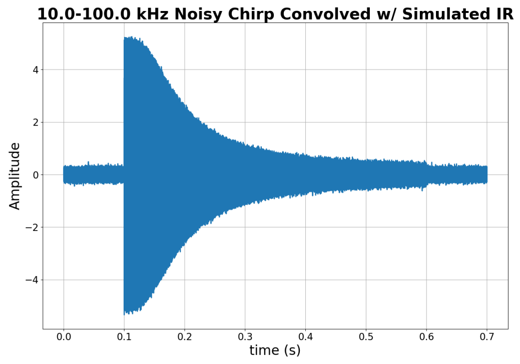


(d) FRF w/ fs=500kHz

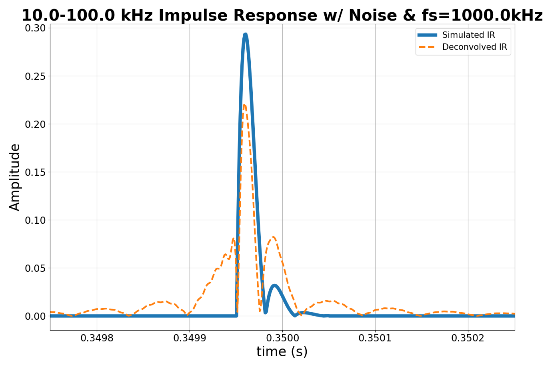
Figure 3.11 Deconvolved impulse and frequency responses for a simulated 10-100 Hz swept-sine signal with various sampling rates (f_s). This figure demonstrates the potential for choosing a sampling rate that is too high for precise results.



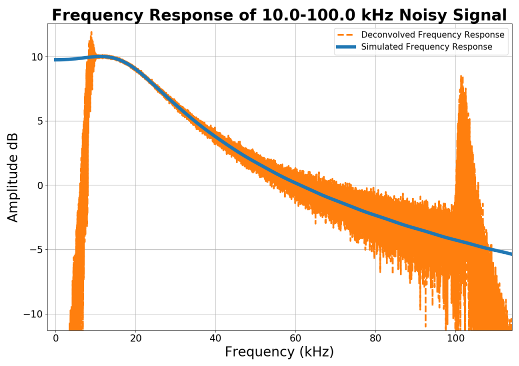
(a) Noisy swept-sine signal



(b) Simulated Recording



(c) Noisy IR



(d) Noisy FRF

Figure 3.12 Deconvolved impulse and frequency responses of a noisy 10-100kHz swept-sine signal with simulated values in blue and deconvolved values in orange.

frequency response, which are shown as orange lines in (c) and (d). Comparisons with the simulated impulse response and frequency response (blue lines) show evidence of noise in the estimated responses: The estimated frequency response function has noise but maintains the expected trend, and the impulse response shows little to no obvious impact from the noise compared to the data without noise (seen in Fig. 3.9b and 3.8b).

This method of acquiring the impulse response via frequency deconvolution of a swept-sine signal has proven effective on simulated data and is used for determining the impulse response and frequency response for both the *in situ* calibration and estimation of the combined response of all transducers in the measurement chain. The frequency deconvolution method is also used in conjunction with the *in situ* calibration to obtain the impulse response of the tank environment at various positions.

3.2.2 *In situ* Calibration

The impulse response is not solely a factor of sound propagation in the water tank. In practice, $h(t)$ also includes the effects of the response of the transducers, including Digital to Analog (D-A) and Analog to Digital (A-D) components on the signal. The contribution of these components to $h(t)$ can ideally be accounted for by application of individual calibrated responses of each component of the measurement chain (shown in Fig. 2.6). Alternately, the contribution of all components may be accounted for by understanding the total through-the-sensor (TTS) response [27, 28, 30, 35, 37]. The technique for obtaining the TTS response relies on an *in situ* calibration to obtain $h(t)$ for the measurement system. The TTS response, $h_{\text{TTS}}(t)$, can be obtained from the calibration measurement via the deconvolution method and then used as a filter (in deconvolution) to obtain the impulse response corresponding to sound propagation in the water tank.

The first step to obtaining the TTS response is taking a calibration measurement, where source and receiver are positioned close enough that transmission losses are reduced significantly and reflections are easily removed (see Fig. 3.13). The small transmission losses during these calibration measurements are assumed negligible in this study; however, a phase adjustment accounting for the small propagation distance is applied. The small distance must be chosen with care because a separation distance that is too short relative to a wavelength may introduce nearfield effects. When appropriate, the separation distance should be chosen to be large compared to a wavelength over the frequency bandwidth of interest. Otherwise, these potential nearfield effects must be noted.

The calibration signal is a swept-sine signal, spanning the bandwidth of interest, which is then broadcast and recorded. This calibration measurement is interpreted through the above

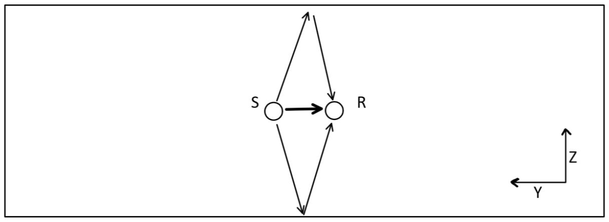


Figure 3.13 Ray paths of first side wall reflections in the water tank with source and receiver positioned close such that propagation losses are reduced significantly. Not to scale.

deconvolution method (described by Eqs. 3.12, 3.13, 3.14), and the resulting impulse response is time-gated for removal of all reflections in order to estimate the response of the measurement chain. This TTS calibration thus incorporates the unit-less sensitivities of all unknown components [7,27]. Application of this response to a subsequent measurement yields a calibrated measured response in Volts (which may be converted to μPa when a transducer sensitivity is applied in the preamplifier settings or directly to the data).

The impulse response obtained through frequency deconvolution [27,35,37] of the calibration measurement is time-gated using a half-Hanning window, as mentioned above in Sec. 3.1.3 but inthis case the time is chosen to remove all reflections. The whole D-A and A-D measurement chain frequency response $H_{\text{TTS}}(f)$ can then be obtained by applying a fast Fourier transform (FFT) on the windowed, time-gated response $h_{\text{TTS}}(t)$. An example of the TTS response in the frequency domain, from a 10-100 kHz signal, is shown in Fig. 3.14. The precision of the TTS response appears to extend beyond the bandwidth of the signal to approximately 150 kHz.

After the TTS response $h_{\text{TTS}}(t)$ is obtained from the calibration measurement, it can be applied to the received signal $r(t)$ to estimate the impulse response of the sound propagation in the environment, $h(t)$, may be found for subsequent measurements. The TTS response is applied as an inverse filter [37] via the classical method as

$$r(t) = h(t) * h_{\text{TTS}}(t) * g(t), \tag{3.16}$$

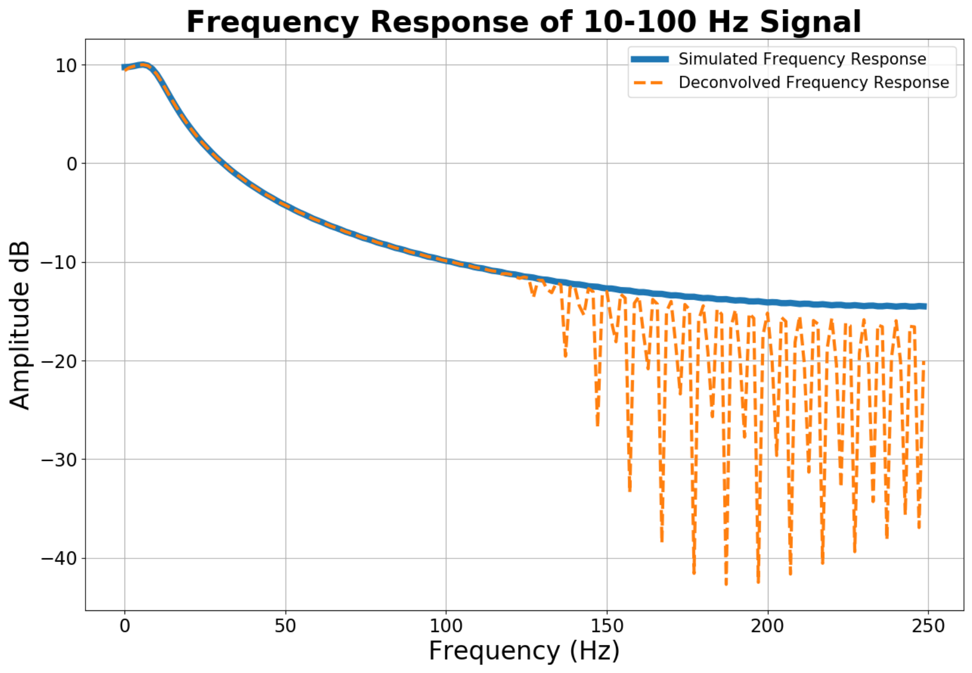


Figure 3.14 Calibration Frequency Response Function (FRF) of a 10-100 Hz swept-sine signal.

where $g(t)$ is the generated signal. After a Fourier transform and rearranging, the frequency response (also referred to as the transfer function) associated with sound propagation in the tank is

$$H(f) = \frac{R(f)}{H_{\mathrm{TTS}}(f)G(f)}, \tag{3.17}$$

or by applying Wiener deconvolution as in Eq. 3.18 in order to prevent division by zero:

$$H(f) = \frac{[H_{\mathrm{TTS}}(f)G(f)]^{*}R(f)}{[H_{\mathrm{TTS}}(f)G(f)]^{2} + \sigma^{2}}. \tag{3.18}$$

An IFFT then yields the impulse response associated with the transfer function of the sound propagation in the tank independent of the frequency response of all the components in the measurement chain:

$$h(t) = F^{-1}[H(f)]. \tag{3.19}$$

Thus, the frequency-dependent calibrated water-tank response can be obtained *in situ* for any source-receiver position within the tank under any propagation conditions. Since acoustic propagation models must adjust for varying conditions such as water temperature gradients [11], this *in situ* calibration and measurement method provides the ability to obtain large data sets with full acoustic characterization within a water tank accounting for varying conditions. A full tank acoustic propagation model may be developed from this methodology.

3.3 Water Tank Models

An effective model is required for acoustic propagation within a water tank enclosure to enable the future work discussed in Sec. 5.1. Under consideration are classical tank models based on idealized boundary conditions, methods that seek to improve the boundary condition of the classical models, and two well accepted two-dimensional ocean acoustic propagation models for comparison.

These models assume a monopole point source with point receiver and time-harmonicity. Particular consideration is given to the importance of the impact of lining the tank with acoustically absorptive panels (as described in Sec. 2.2), on how well the measurements match the models, and to find the frequency band over which propagation in the tank is approximately the same as scale-model ocean propagation.

3.3.1 Helmoltz Equation

The normal-mode eigenfunction solution for a rectangular parallelepiped enclosure satisfies the homogeneous Helmholtz equation under time-harmonic conditions, $e^{j\omega t}$:

$$\nabla^2 \hat{p}(x,y,z) + k^2 \hat{p}(x,y,z) = 0, \tag{3.20}$$

where $\hat{p}$ is the pressure, k is the separation constant, and x, y, and z indicate location in the tank relative to the front, bottom corner of the tank. The k values that correspond to nontrivial solutions to the boundary value problem are the modal eigenvalues of the system: $k_N = (\omega_N/c) = (2\pi f_N/c)$, when N corresponds to the mode number, and the f_N are referred to as the eigenfrequencies. Equation 3.20 can be written in terms of these modal eigenvalues and mode functions $\Psi_N(x,y,z)$ as

$$\nabla^2 \Psi_N(x,y,z) + k_N^2 \Psi_N(x,y,z) = 0. \tag{3.21}$$

The mode functions are orthogonal:

$$\iiint_V \Psi_N \Psi_{N'} dV = \begin{cases} 0;\ N \neq N' \\ V\Lambda_N;\ N = N' \end{cases} \tag{3.22}$$

Due to this orthogonality relation, where V is the volume of the tank and Λ_N are the mean values of $|\Psi_N|^2$, the Green's function for propagation between position $r = (x,y,z)$ and $r_0 = (x_0,y_0,z_0)$ is

$$G_\omega(r|r_0) = \frac{-4\pi}{V} \sum_{N=0}^{\infty} \frac{\Psi_N(r_0)\Psi_N(r)}{(k^2 - k_N^2)\Lambda_N}, \tag{3.23}$$

and the pressure in three-dimensional Cartesian coordinates (relative to the tank coordinates with $z = 0$ m at the bottom of the tank) is proportional to the Green's function in Eq. 3.23:

$$\hat{p}(\vec{r}) \approx \hat{A} G_\omega(r|r_0) \tag{3.24}$$

with some amplitude $\hat{A}$.

3.3.2 Neumann & Dirichlet Boundaries

The standing wave solution or overall modal response of an enclosed sound field is a superposition of up to eight possible traveling waves, one into each octant of the three-dimensional coordinate system. A classical approximation used to model a six-walled parallelipiped room enclosure assumes rigid-boundaries satisfying the homogeneous Neumann pressure boundary conditions: The normal component of the spatial derivative of the acoustic pressure, i.e., the acoustic particle velocity, is zero at the boundaries, but the acoustic pressure is nonzero.

Different approaches are available for developing an idealized model for the open-air water tank. One approach is to assume the five walls are rigid with Neumann boundary conditions and treat the water-air interface as a pressure-release boundary, corresponding to a homogeneous Dirichlet boundary condition, where the acoustic pressure is zero but the normal component of the particle velocity is non-zero [3]. A model for water tank enclosures may alternately use pressure-release boundaries on all sides as in Ref. [23,29,53]. However, Li *et al.* [29] determined that their glass tank boundaries could be simulated as pressure-release (Dirichlet boundaries) only when the frequency of interest was lower than the first modal frequency. These results only represent idealized solutions and fail to account for non-idealized impedance characteristics of real tank boundaries particularly at higher frequencies.

For the rigid-walled solution with a pressure-release water-air interface, the eigenvalues are

$$k_x = \frac{n_x \pi}{L_x},\ k_y = \frac{n_y \pi}{L_y},\ k_z = \frac{(2n_z - 1)\pi}{2L_z} \tag{3.25}$$

with the following associated eigenfunctions and eigenfrequencies:

$$\Psi_N = \cos\left(\frac{n_x \pi x}{L_x}\right)\cos\left(\frac{n_y \pi y}{L_y}\right)\cos\left(\frac{(2n_z - 1)\pi z}{2L_z}\right) \tag{3.26}$$

and

$$f_N = \frac{c}{2}\left[\left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2 + \left(\frac{2n_z - 1}{2L_z}\right)^2\right]^{\frac{1}{2}} \tag{3.27}$$

for $N = (n_x, n_y, n_z)$ with $n_x, n_y = 0, 1, 2, ...$ and $n_z = 1, 2, 3, ...$ (since the $n_z = 0$ solution is equivalent to the $n_z = 1$ solution).

Or more specifically, the Green's function for the rigid-walled rectangular parallelepiped room with a monopole at the source position $r_0 = (x_0, y_0, z_0)$ and a point receiver at field positions $r = (x, y, z)$ is

$$G_\omega(r|r_0) = \frac{-4\pi}{V}\sum_{n_x=0}^{\infty}\sum_{n_y=0}^{\infty}\sum_{n_z=0}^{\infty}\frac{\Psi_N(x_0, y_0, z_0)\Psi_N(x, y, z)}{(k^2 - k_N^2)\Lambda_N}, \tag{3.28}$$

where $\Lambda_N = \frac{1}{\varepsilon_{n_x}\varepsilon_{n_y}\varepsilon_{n_z}}$ with the Neumann factor

$$\varepsilon_{nx,ny} = \begin{cases} 1;\ i = 0 \\ 2;\ i \neq 0 \end{cases}$$

$$\varepsilon_{nz} = 2$$

.

The alternate assumption of a pressure-release boundary (Dirichlet) condition on all six sides is similar but has eigenvalues $k_x = \frac{n_x \pi}{L_x}$, $k_y = \frac{n_y \pi}{L_y}$, $k_z = \frac{n_z \pi}{L_z}$ with $n_x, n_y, n_z = 1, 2, 3, ...$ and associated eigenfunctions and eigenfrequencies of

$$\Psi_N = sin\left(\frac{n_x \pi x}{L_x}\right) sin\left(\frac{n_y \pi y}{L_y}\right) sin\left(\frac{n_z \pi z}{L_z}\right) \tag{3.29}$$

and

$$f_N = \frac{c}{2}\left[\left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2 + \left(\frac{n_z}{L_z}\right)^2\right]^{\frac{1}{2}}. \tag{3.30}$$

By applying the orthogonality conditions to this solution, the Green's function is

$$G_\omega(r|r_0) = \frac{-4\pi}{V}\sum_{nx=0}^{\infty}\sum_{ny=0}^{\infty}\sum_{nz=0}^{\infty}\frac{\sin\left(\frac{nx\pi x_0}{L_x}\right)\sin\left(\frac{ny\pi y_0}{L_y}\right)\sin\left(\frac{nz\pi z_0}{L_z}\right)\sin\left(\frac{nx\pi x}{L_x}\right)\sin\left(\frac{ny\pi y}{L_y}\right)\sin\left(\frac{nz\pi z}{L_z}\right)}{(k^2-k_N^2)\Lambda_N} \tag{3.31}$$

where $\Lambda_N = \Lambda_{nx,ny,nz} = \frac{1}{\varepsilon_{nx}\varepsilon_{ny}\varepsilon_{nz}}$ with $\varepsilon_i = 2$.

However, since the walls of a water tank are neither perfectly rigid nor perfectly pressure-release, these solutions are of limited use as they do not account for finite-impedance boundary conditions, which are discussed in the Sec. 3.3.3.

3.3.3 Robin Boundary

The solution is obviously more challenging when mixed, or Robin boundary conditions exist. For the tank, Robin boundary conditions correspond to lossy-walls or finite impedance boundary conditions. The finite-impedance boundary approach [29,43,53] attempts to account for the specific finite-impedance characteristics of the boundary while still assuming a pressure-release water-air boundary at the surface for open-air water tanks. However, this approach requires accurately measured tank wall characteristics, particularly the frequency-dependent impedance, which is in itself dependent on the absorption of the boundary as discussed in Sec. 3.1.4.

Following the derivation in Pierce's book [43], the homogeneous Robin-boundary condition is

$$\nabla\widehat{p}(\overrightarrow{r_s})\cdot\overrightarrow{n}_{\text{out}} = -jk\left[\frac{\rho_0 c}{z_s(\overrightarrow{r_S})}\right]\widehat{p}(\overrightarrow{r_s}). \tag{3.32}$$

In this equation, $\vec{n_{\text{out}}}$ indicates the direction perpendicular to the walls, z_s, the finite impedance of surface s, ρ_0 and c are respectively the density and sound speed of the medium in the enclosure, and $k = \omega/c$ is the acoustic wavenumber. Also, $\widehat{p}$ is the complex amplitude of acoustic pressure, and $\vec{r_S}$ is a vector position at the boundary surface. This boundary condition can be used to obtain a modal model for an enclosure with walls of finite impedance z_s. Using the completeness property, an expansion can be found for $\widehat{p}(\vec{r})$ in terms of the Ψ_N appropriate for the rigid-wall boundary conditions case in Eq. (3.26):

$$\hat{p}(\vec{r}) \approx -4\pi\hat{A} \sum_{N=0}^{\infty} \frac{\Psi_N(\vec{r_0})\Psi_N(\vec{r})}{V\Lambda_N \left\{k^2 - k_N^2 - jk\left[\frac{1}{V\Lambda_N}\iint_S \Psi_N(\vec{r_s})\frac{\rho_0 c}{z_s(\vec{r})}dS\right]\right\}}, \tag{3.33}$$

with $\Lambda_N = \Lambda_{nx,ny,nz} = \frac{1}{\varepsilon_{nx}\varepsilon_{ny}\varepsilon_{nz}}$ and

$$\varepsilon_{nx,ny} = \begin{cases} 1;\ i = 0 \\ 2;\ i \neq 0 \end{cases}$$

$$\varepsilon_{nz} = 2$$

.

This expression can be simplified to

$$\hat{p}(\vec{r}) \approx -4\pi\hat{A} \sum_{N=0}^{\infty} \frac{\Psi_N(\vec{r_0})\Psi_N(\vec{r})}{V\Lambda_N \left\{k^2 - k_N^2 - jk\left[\frac{<\alpha>_S S}{4V}\right]\right\}} \tag{3.34}$$

by defining an effective, spatially averaged, absorption term $<\alpha>_S$:

$$\frac{<\alpha>_S S}{4V} = \frac{1}{V\Lambda_N}\iint_S \Psi_N(\vec{r_s})\frac{\rho_0 c}{z_s(\vec{r})}dS. \tag{3.35}$$

However, this solution does not account for the thermoviscous and molecular-relaxation losses as the wave propagates within the medium, which introduce complex eigenvalues. Even greater accuracy may be accomplished with a model accounting for these propagation losses [45–47] using the imaginary part of the modal eigenvalues, as discussed in Sec. 3.1.1. Because the fluid and walls include intrinsic damping mechanisms, the acoustic standing waves have nonzero nodes, damped natural frequencies, and they decay over time. As the model improves to include these effects, greater efficacy may be obtained.

As mentioned, a propagating-wave solution to account for thermoviscous and molecular relaxation losses within the medium introduces a complex wavenumber $\tilde{k}$. This complex eigenvalue can be separated into $\tilde{k}_N = \frac{\tilde{\omega}_N}{c} = \frac{\omega_N}{c} + \frac{j\delta_N}{c} = k_N + j\alpha_N$, where α_N is modal absorption and δ_N is the modal damping factor determined from the T_{60} as

$$\delta_N(f) = -\frac{\ln(10^{-6})}{2T_{60}(f)}. \tag{3.36}$$

This damping effect is approximated in the finite impedance model.

The propagation losses are further applied to our solution by calculating the spatially averaged absorption coefficient of the walls (Eq. 3.37) using the estimated propagation losses when solving the modified Norris-Eyring equation (Eq. 3.2) from the measured T_{60}. The resulting spatially averaged absorption is

$$<\alpha>_S = 1 - \exp\left[\frac{8V<\alpha>_p}{S} - \frac{24\ln 10}{cT_{60}}\right] \tag{3.37}$$

where $<\alpha>_p$ is the propagation absorption coefficient, V is the enclosure volume, S is the total surface area of the boundaries, c is the speed of sound, and T_{60} is the reverberation time.

This approximate solution can be used as a model, alongside both the Neumann & Dirichlet models. This finite-impedance model is referred to in Ch. 4 as the "Pierce" model and is compared alongside the model developed in Novak *et al.* 2018 [53], which demonstrated strong validity in smaller tanks assuming pressure-release boundaries and applying finite-impedance to the solution. These Cartesian models are compared to measurements in Ch. 4 and to results from open-ocean models, such as ORCA discussed in Sec. 3.3.4.

3.3.4 Ocean Model

The field of underwater acoustics focuses on sound-wave phenomena in the ocean and seabed. Modeling underwater sound propagation is a large endeavor. Many of these models are open-source and part of the Ocean Acoustics Library at https://oalib-acoustics.org . The models are grouped according to the methodology used and the assumptions made: ray-tracing, normal-modes, parabolic equation solver, and wavenumber integration.

The first ocean acoustics model that has been used by Dr. Neilsen's group is called ORCA [15]. ORCA is a range-independent, normal-mode model that computes solutions to the wave equation in cylindrical coordinates with the assumption of azimuthal symmetry (i.e., no lateral reflection). For the specified frequency f, ORCA computes the normalized mode functions, $\bar{\phi}_n(z)$, and modal eigenvalues, k_n, as a function of frequency based on how the sound speed, density, and absorption vary as a function of depth. A modal summation is used to obtain the Green's function corresponding with a point source at depth z_s and range r from a receiver at depth z. This is expressed in Eq. (15) of Westwood *et al.* [15] in terms of normalized mode function as

$$p(r,z) = \frac{i\pi}{\rho_s}\sum_n \bar{\phi}_n(z_s)\bar{\phi}_n(z)H_0^{(1)}(k_n r), \tag{3.38}$$

where ρ_s is the density at z_s, and the $H_0^{(1)}$ is the zeroth-order Hankel function of the first kind. When $|k_n r|$>5, the asymptotic expression is used and the Green's function becomes

$$p(r,z) = \sqrt{2i\pi}e^{i\pi/4}\frac{1}{\rho_s}\sum_n \frac{\bar{\phi}_n(z_s)\bar{\phi}_n(z)e^{ik_n r}}{\sqrt{k_n r}}, \tag{3.39}$$

In this manner, ORCA can produce the frequency response of a water waveguide with depth-dependent sound speed and a horizontally stratified seabed. The validity of the frequency response depends on how well the parameterization of the acoustic properties of the environment match reality. This normal-mode model is highly robust, accounting for a wide range of real world characteristics such as leaky boundaries(surface and seabed) and does so with low computational costs. The $\frac{1}{\sqrt{r}}$ cylindrical spreading follows a far-field approximation common in open-ocean modeling and may not be the best representation of the zeroth-order Hankel function for this tank where a $\frac{1}{r}$ dependence may more appropriately account for top and bottom reflection and no far-field assumption. This idea needs to be evaluated. However, further development of the ORCA model for the tank is beyond the scope of this thesis. ORCA is commonly used as an acceptable open-ocean model or benchmark solution and for this reason is of interest here.

In Ch. 4, the frequency response function for a tank experiment is modeled with ORCA and compared, alongside the finite-impedance models, to that obtained from measurements using the methods described in Sec. 3.2.1. This comparison provides an indication of how well sound propagation in the tank reflects the azimuthal boundary assumption often assumed in ocean acoustics models.

3.3.5 Anechoic Lining for Improved Modeling

While ORCA and most open-ocean models assume azimuthal symmetry, the tank enclosure is bounded by side walls. To more closely match the tank environment to an azimuthally symmetric model, the side reflections need to be removed or attenuated. One way to remove lateral reflection during processing is by time-gating, however this will also remove signals associated with additional top and bottom bounces, which should be included in a scale-model ocean setup. The decision was made to reduce the lateral reflections by lining the side walls with with the application of anechoic panels (Apltile SF5048 acoustic attenuating material from Precision Acoustics). The side-wall reflected acoustic energy is significantly decreased as discussed in Sec. 3.1.3 and seen in Fig. 3.1. Further evaluation of the added absorption of the panels along with the acrylic walls was performed with the hope that the characterization of the added wall absorption would be applied to the model via the spatially averaged absorption and more closely match the azimuthally-symmetric open-ocean model represented by ORCA. This assessment of the spatially averaged absorption coefficient of the anechoic walls of the tank follows the methods discussed above in Sec. 3.1.4 for determining the absorption of the acrylic walls. Applying the absorption characteristics of both the acrylic and the anechoic panels to the finite-impedance model should bring this modal model, with applied anechoic walls, into closer agreement with the ORCA model for open-oceans. An initial assessment of these models in comparison to the ORCA model is presented in Sec. 4.3.

3.4 Conclusion

Since measurements using a swept sine source have proven to be repeatable and provide a high SNR, swept-sine measurements are taken for many source-receiver positions throughout the tank. The tank dimensions limit the effective tank bandwidth to 5 kHz and above (which is well above the Schroeder frequency for this depth of water). Hence, calculations for obtaining the frequency response or Green's function within the tank are done over the 10-500 kHz band. This response represents the frequency dependent effect the tank has on acoustic propagation with sensor sensitivities accounted for by *in situ* calibration techniques discussed in this Chapter. By obtaining effective acoustic characterization of the tank, particularly of boundary absorption, effective acoustic propagation models may be developed using finite-impedance boundary conditions. The comparison between the ORCA-generated, normal mode model-generated and measurement-based frequency responses discussed in Sec. 4.3, yield insights into the best manner for simulating labeled data to train machine learning algorithms in future research.

Chapter 4

Experimental Results

The primary results for this thesis are discussed over four bandwidths of interest (5-10 kHz, 10-50 kHz, 50-100 kHz, 100-500 kHz). For each frequency band, scans were taken with bare acrylic tank walls and again where the side walls are treated with anechoic paneling. For each scan, a consistent water depth of 0.5 m was maintained. Brüel & Kjær 8103 phase matched transducers were used as both source and receiver for bandwidths under 100 kHz and Teledyne Reson TC4038 transducers were used as both source and receiver for the 100-500 kHz bandwidth. Information about the primary datasets is provided in Appendix A.3.

The scans covered 729 different source and receiver position pairs throughout the tank environment as seen in Fig. 4.2. The (x,y,z)source positions formed an evenly spaced 3x3x3 grid from (0.41 m,2.83 m,0.25m) to (0.81 m,2.13 m,0.35m), and receiver positions formed an evenly spaced 3x3x3 grid from (0.41 m,0.83 m,0.25m) to (0.81 m,1.83 m,0.35m). This large grid allowed for the assessment of sound propagation throughout the tank and provided more than enough averaging to obtain the reverberation time for characterization of the tank boundaries effectively through reverse Schroeder integration [49] (Sec. 3.1.2) of the impulse response (Sec. 3.2.1). Results are presented following the work flow seen in Fig. 4.1.

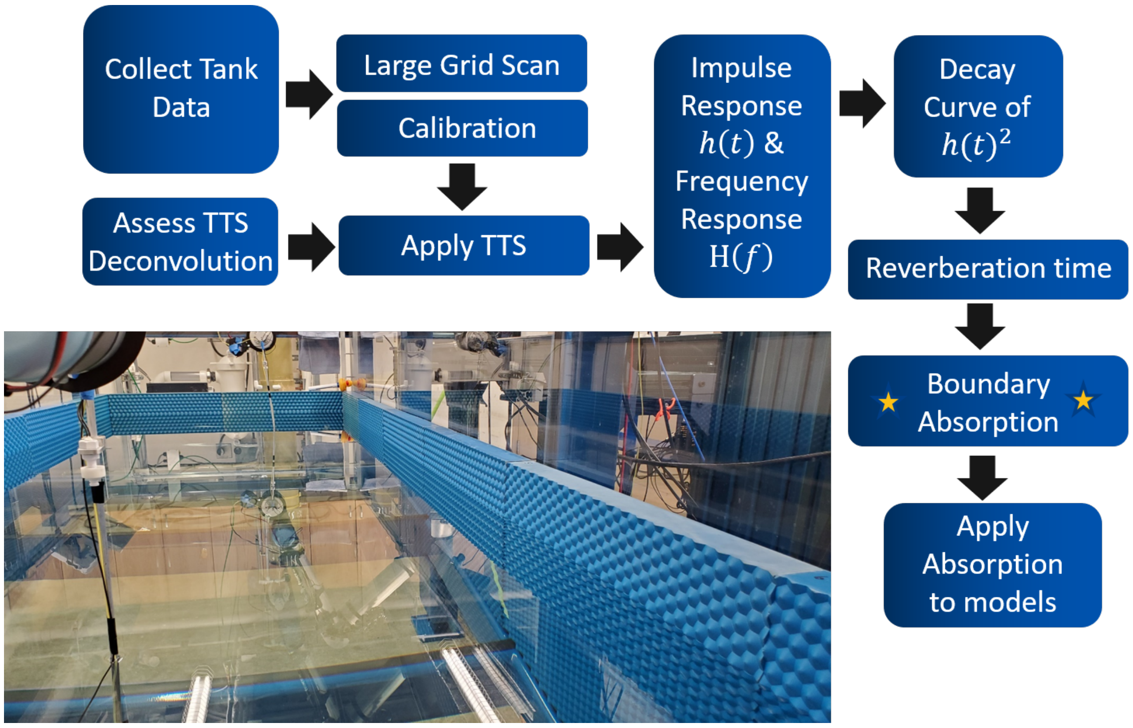


Figure 4.1 Work flow to obtain boundary characterization results of the water tank environment.

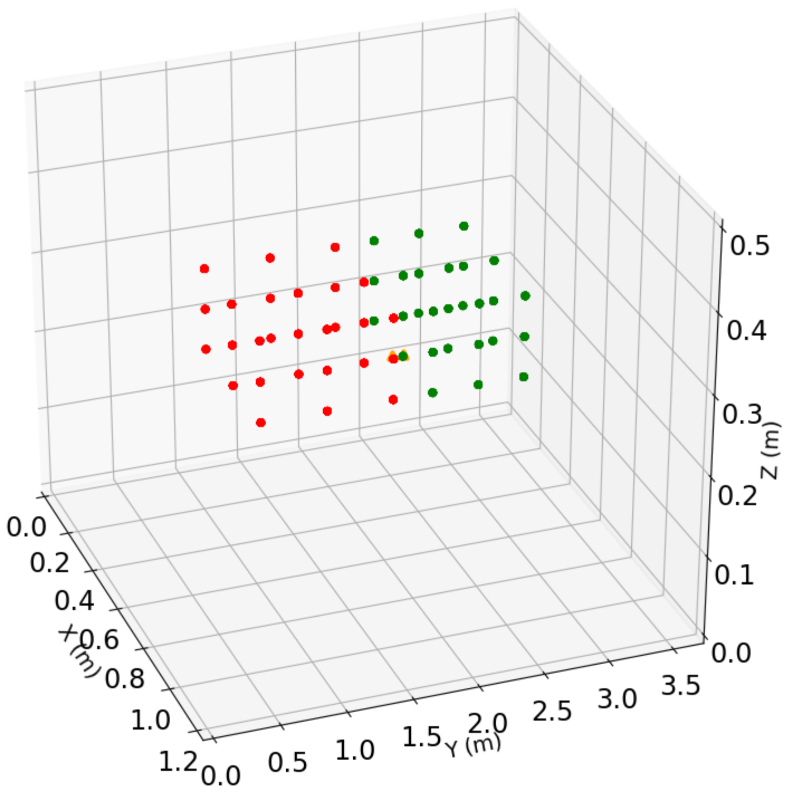


Figure 4.2 Scan grid within the tank environment. Red positions represent the source positions (using the UR10e Ægir) and green represent the receiver positions (using the other UR10e Rán). Calibration positions are not displayed but are located in the center of the grid at (0.6 m,2.14 m,0.25m) and (0.6 m,2.06 m,0.25m).

4.1 Through the Sensor Response

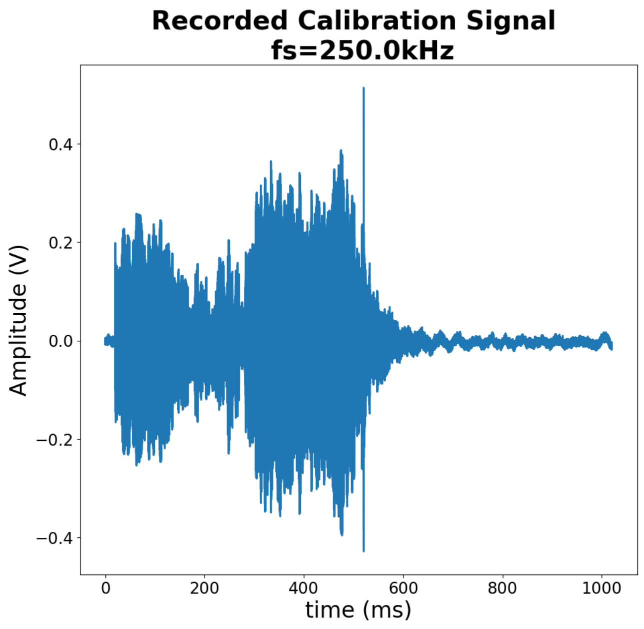
The TTS response (Sec. 3.2.2) is obtained from a calibration measurement where tank environmental effects are time-gated out, as discussed in Sec. 3.1.3 with the algorithm shown in Appendix B.2.1. This process provides a frequency response representation of the measurement chain. This response $H(f)$ may be used to deconvolve the effects of the measurement chain from subsequent measurements and obtain an impulse response $h(t)$ and frequency response $H(f)$ of the sound propagation in the tank environment without the effects of all components of the measurement chain on a measurement.

4.1.1 Calibration Measurements

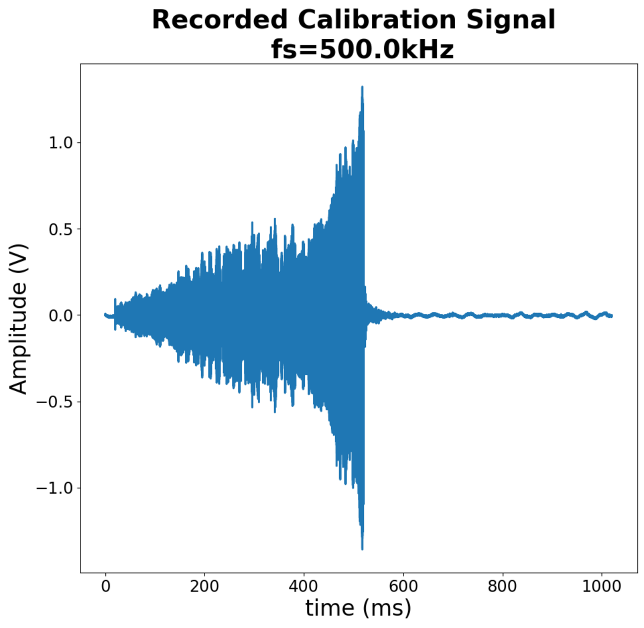
Calibration measurements are taken in the manner discussed in Sec. 3.2. Examples of recorded calibration measurements in the plain acrylic-walled tank are shown in Figs. 4.3 and 4.4 and for the tank treated with anechoic panels in Figs. 4.5 and 4.6. Each figure displays the raw recording of a swept-sine signal in the tank at the calibration positions (0.6 m,2.14 m,0.25m) and (0.6 m,2.06 m,0.25m) for a 5-10 kHz, 10-50 kHz, 50-100 kHz, and 100-500 kHz swept-sine signal. Notice the initial shape for each bandwidth is similar between the acrylic-walled and anechoic-walled calibrations but as time goes on the shape is altered due to increased absorption of side wall reflections. These signals are processed through frequency deconvolution into impulse response measurements, which are then time-gated.

4.1.2 Calibrated Impulse Response

The impulse response of each calibration measurement is obtained through frequency deconvolution according to Eq. 3.13 and 3.14 (see also Appendix B.2.3). Examples of the calculated impulse response, $h(t)$, are shown in Fig. 4.7 for tank calibration measurements with acrylic walls. Each

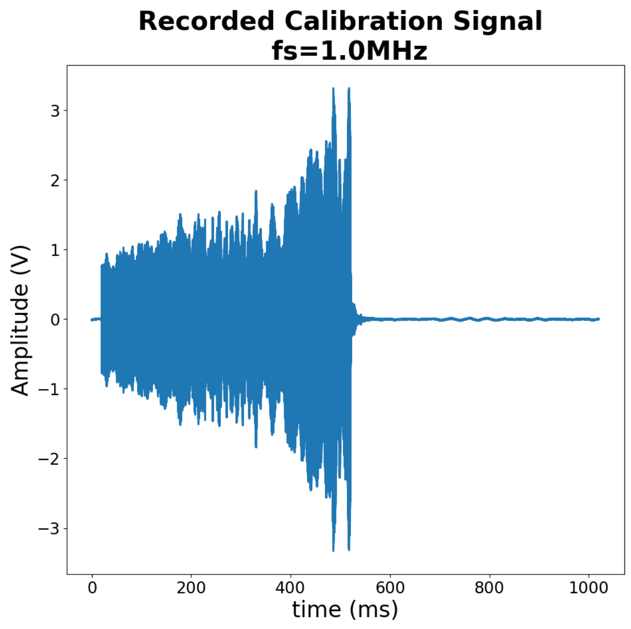


(a) 5-10kHz Acrylic

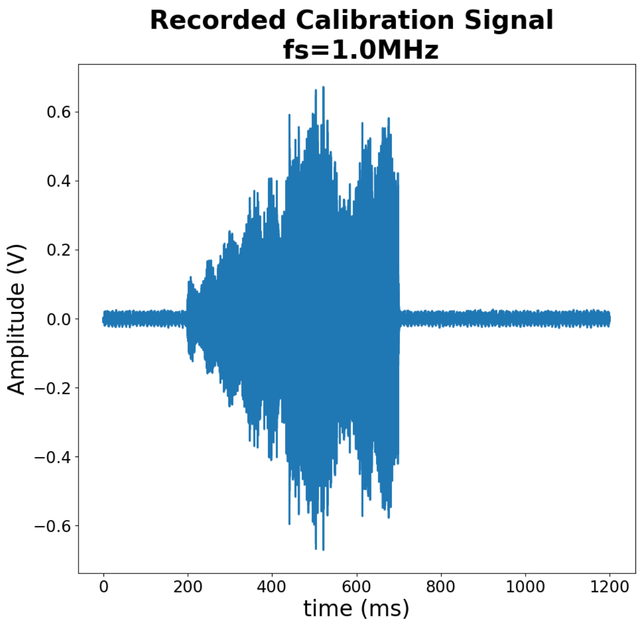


(b) 10-50kHz Acrylic

Figure 4.3 Raw recorded time waveforms of calibration measurements by bandwidth for tank with acrylic walls and no anechoic panels for the 5-10 kHz and 10-50 kHz bandwidths.

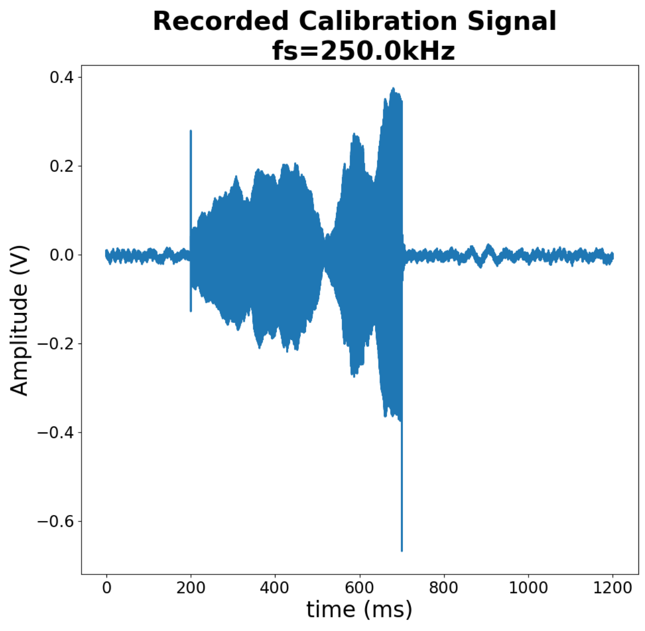


(a) 50-100kHz Acrylic

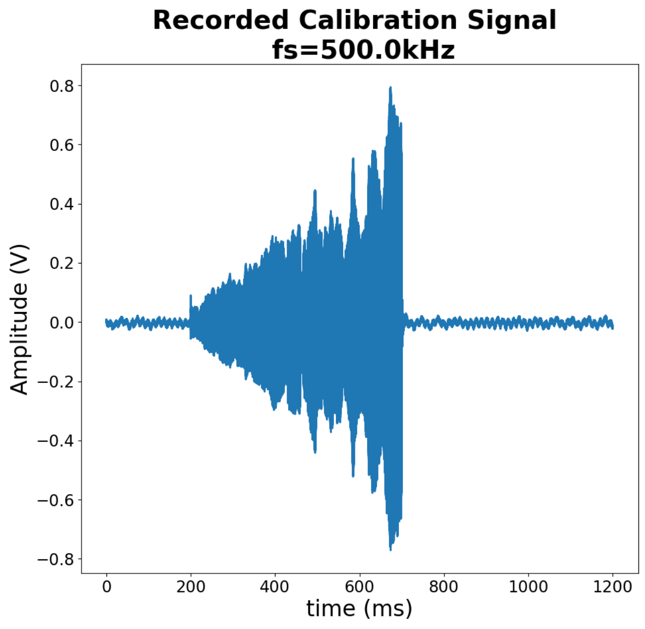


(b) 100-500kHz Acrylic

Figure 4.4 Raw recorded time waveforms of calibration measurements by bandwidth for tank with acrylic walls and no anechoic panels for the 50-100 kHz and 100-500 kHz bandwidths.

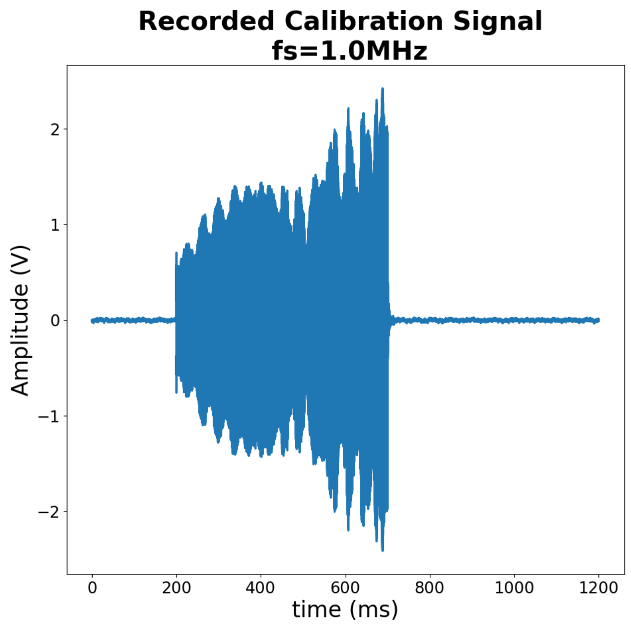


(a) 5-10kHz Anechoic

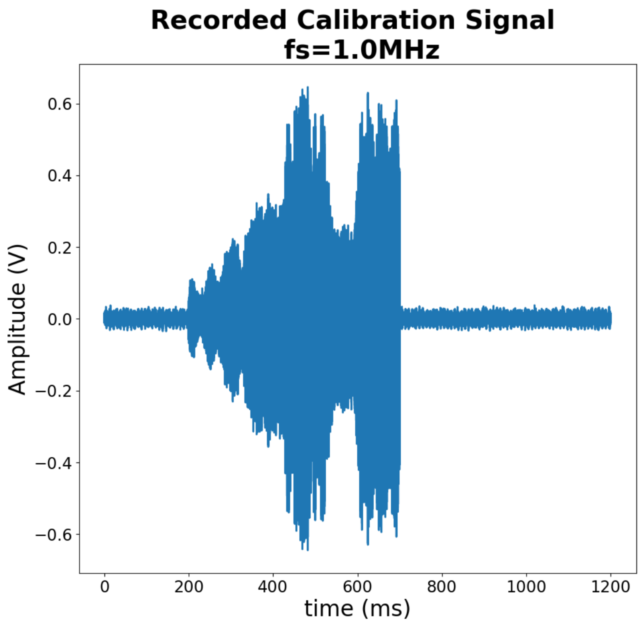


(b) 10-50kHz Anechoic

Figure 4.5 Raw recorded time waveforms of calibration measurements for swept-sine signals over various frequency bands for tank with acrylic walls covered by anechoic panels for the 5-10 kHz and 10 kHz-50 kHz bandwidths.



(a) 50-100kHz Anechoic



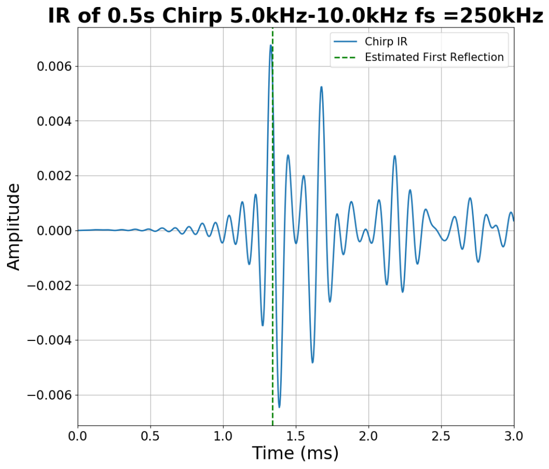
(b) 100-500kHz Anechoic

Figure 4.6 Raw recorded time waveforms of calibration measurements for swept-sine signals over for the 50-100 kHz and 100-500 kHz bands for tank with acrylic walls covered by anechoic panels.

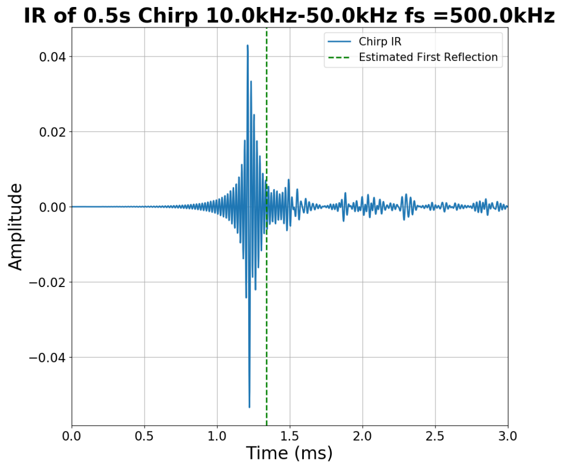
impulse response is time-gated according to predicted reflection arrival times (seen as green dashed lines in the plots) to remove reflections. The results of the time-gated impulse response should be the same for both the acrylic-walled tank and the anechoic-walled tank. The Fourier transform of the time-gated impulse response is the TTS frequency response $H_{\mathrm{TTS}}(f)$. (See examples in Fig. 4.8.) These frequency responses account for the frequency-dependent effects the measurement chain has on a signal and can be applied as a calibration through deconvolution to subsequent measurements to obtain the transfer function, or frequency response of the sound propagation for any given source-receiver pair positions. It is important to note that there is a significant difference between the 100 kHz response due to the different transducers used for the 50-100 kHz and 100-500 kHz bands (as discussed in Appendix A.3) have on the measurement chain. Therefore, these TTS responses may be used effectively to calibrate measurements taken with those specific transducers. The transfer function essentially defines how acoustic energy propagates in the tank environment from a specific source to a specific receiver.

4.1.3 Tank Transfer Function

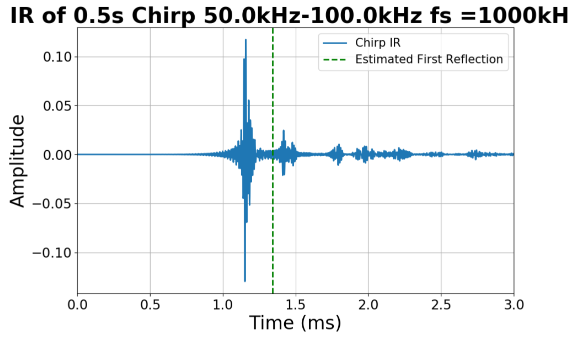
The transfer function, also known as the frequency response function $H(f)$, describes the sound propagation in the tank environment and, therefore, provides an estimate of the frequency-dependent transmission loss (TL), or total reduction in signal intensity, of acoustic energy due to the acoustical properties of the medium and boundaries of the enclosed environment. As an example, Figs. 4.9 and 4.10 show the impulse response, as well as the frequency response of the transfer function with the anechoic panels lining the walls. The responses for each bandwidth were obtained from measurements with the same source and receiver positions of (0.610 m,2.130 m,0.300 m) and (0.610 m,1.830 m,0.300 m), respectively. The frequency response functions show large background noise below 5 kHz and elevated levels over the signal bandwidth. The impulse responses (IR) show peaks representing various acoustic energy reflections within the enclosure with increasing



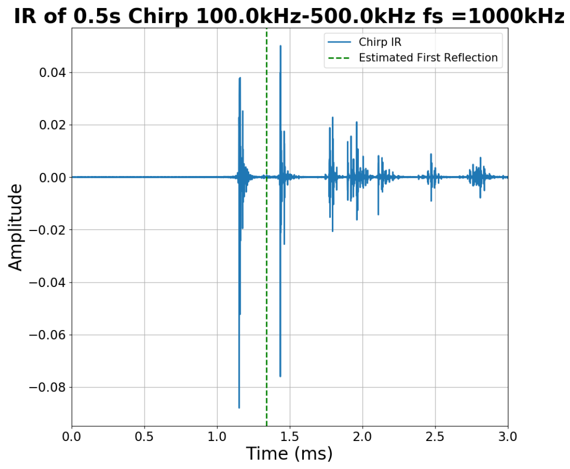
(a) 5-10kHz



(b) 10-50kHz

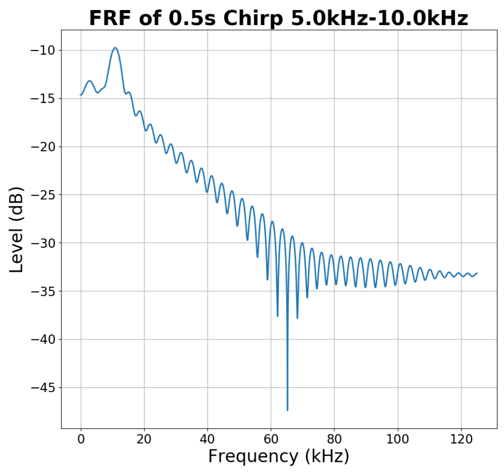


(c) 50-100kHz

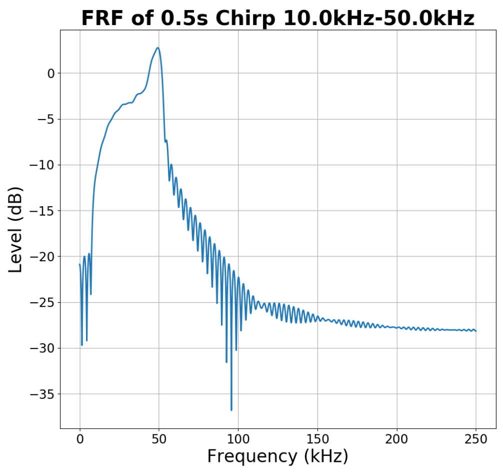


(d) 100-500kHz

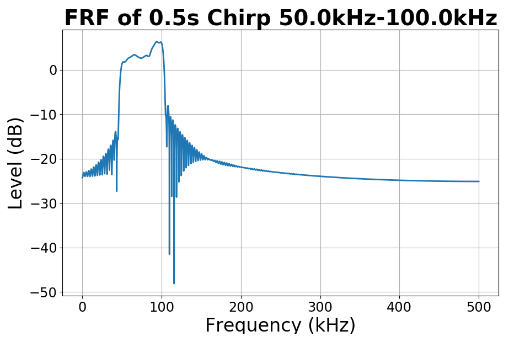
Figure 4.7 Deconvolved impulse responses of measured swept-sine calibration signals showing predicted time of arrival for a first reflection where time-gating would occur.



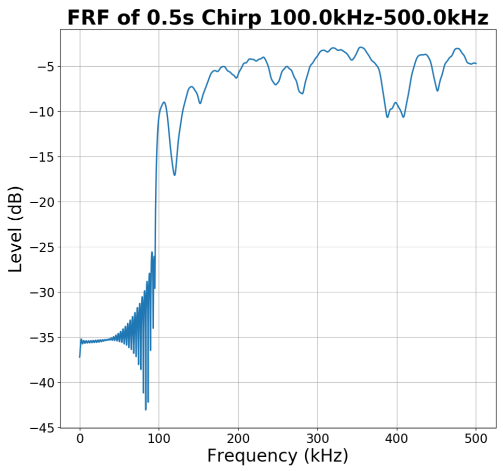
(a) 5-10kHz



(b) 10-50kHz

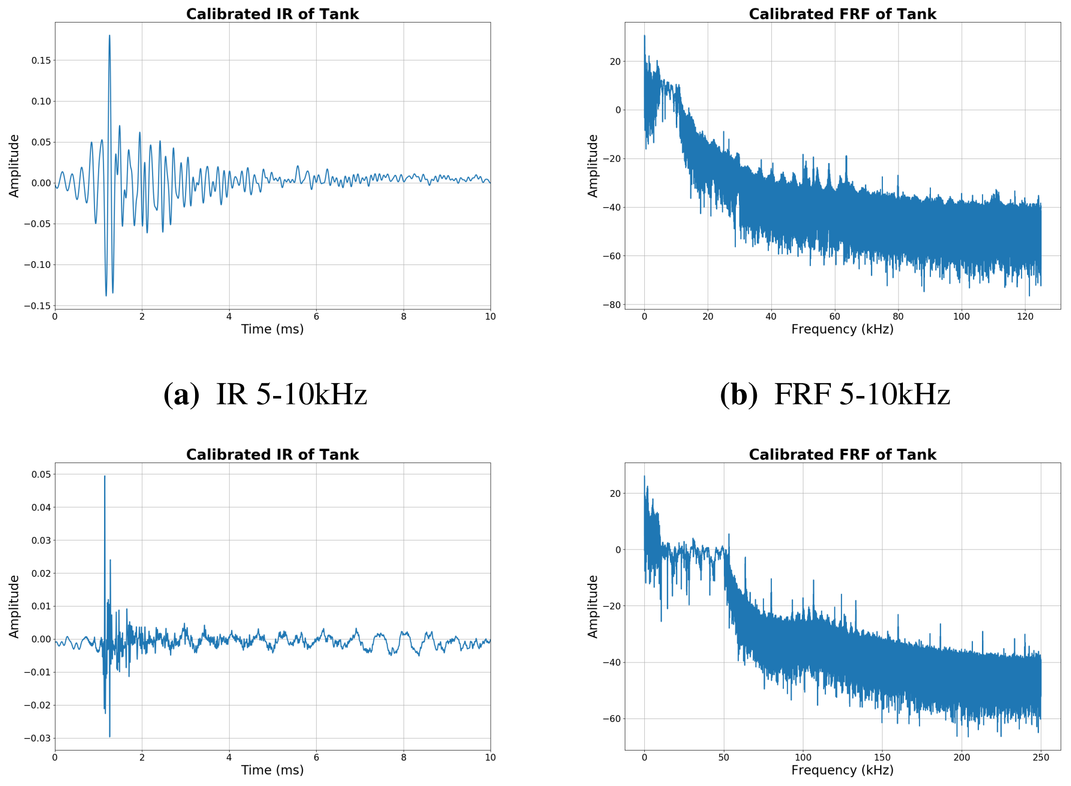


(c) 50-100kHz



(d) 100-500kHz

Figure 4.8 Frequency response of time-gated calibration measurements is the TTS response of the measurement chain. Each TTS response is deconvolved from subsequent measured signals to obtain a pure tank response with effects caused by the measurement chain removed.



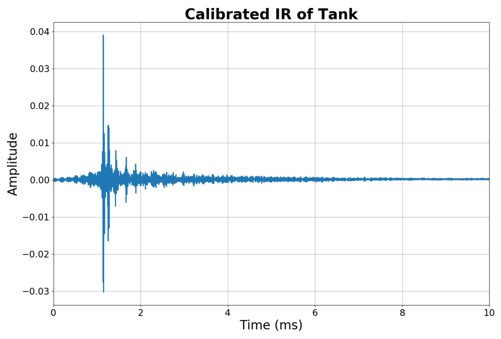
(a) IR 5-10kHz **(b)** FRF 5-10kHz

(c) IR 10-50kHz **(d)** FRF 10-50kHz

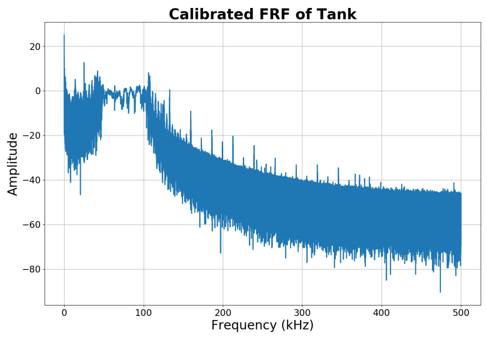
Figure 4.9 Deconvolved impulse and frequency responses for measured swept-sine signals (5-10 kHz and 10-50 kHz) at source (0.610 m, 2.130 m, 0.300 m) and receiver (0.610 m, 1.830 m, 0.300 m) positions with anechoic panels lining the walls of the tank.

transmission loss. The frequency responses demonstrate the efficiency of various frequencies propagating through the enclosure and show how certain frequencies propagate more efficiently than others as seen by higher amplitudes peaks at those frequencies. These responses define how acoustic energy propagates from the specific source position to the receiver position within the environment and are key to understanding the how much energy is absorbed by the boundaries.

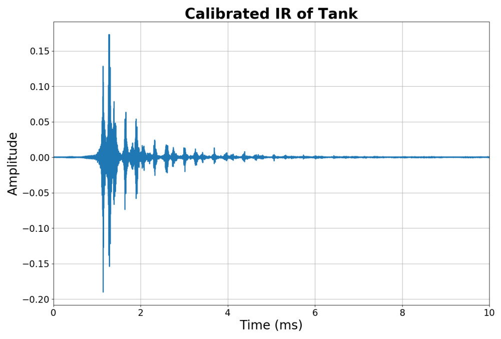
The effect on the response of source-receiver position is now considered for the 10-50 kHz band for measurements taken without anechoic panels on the walls. First, the impulse and frequency responses as the source-receiver range increases down the length of the tank are shown in Fig. 4.11.



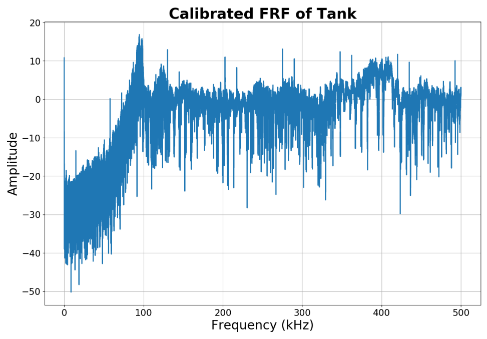
(a) IR 50-100kHz



(b) FRF 50-100kHz



(c) IR 100-500kHz



(d) FRF 100-500kHz

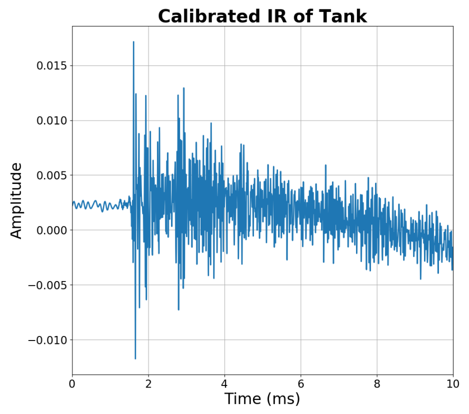
Figure 4.10 Deconvolved impulse and frequency responses for measured swept-sine signals (50-100 kHz and 100-500 kHz) at source (0.610 m, 2.130 m, 0.300 m) and receiver (0.610 m, 1.830 m, 0.300 m) positions with anechoic panels lining the walls of the tank.

The frequency response functions look nominally similar as the source-receiver distance increases with some slight variation by frequency. Notice a more significant DC offset seen for each impulse response demonstrating the position dependence of the response as range increases. The changes in the responses at different positions across the width of the tank is shown in Fig. 4.12. The three positions correspond to positions on one side of the tank, the middle of the tank and the opposite side. Similarly, the effect of acoustic propagation with increased depth in the tank may be assessed in Fig. 4.13. For each of these figures, only one Cartesian dimension was varied at a time. As mentioned, small changes can be seen in the frequency response functions, but the overall shape remains similar. The impulse responses show a more clear position dependence within the tank environment along with an interesting changing DC offset for some positions.

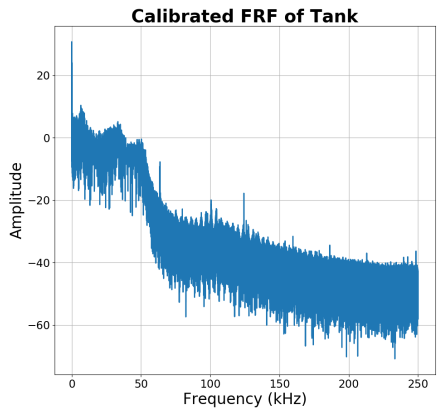
Only slight similarities, differences and trends lend themselves to understanding how sound propagates through the tank environment from these figures alone. However, the position-dependent frequency response of the tank estimates the TL as sound propagates and is a result of the acoustic properties of the environment. Thus, these frequency responses allow for calculating those parameters, such as acoustic absorption of the environmental boundaries. Understanding how to properly characterize the boundary conditions of an environment allows for more effective modeling of the environment. An initial evaluation of models with applied boundary conditions is discussed in Sec. 4.3.

4.2 Finite-impedance modeling

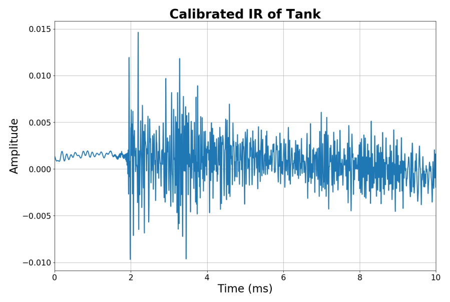
An important step to enabling finite-impedance modeling is to obtain an estimate of the reverberation time T_{60}. According to standards ISO 354:2003 [50], ISO3382-1:2009 [41], and ISO3382-2:2008 [42], T_{60} should ideally be obtained from impulse response measurements at multiple random positions within the enclosed environment. An averaged T_{60} is obtained by determining the decay



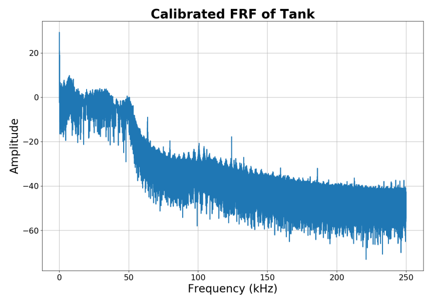
(a)



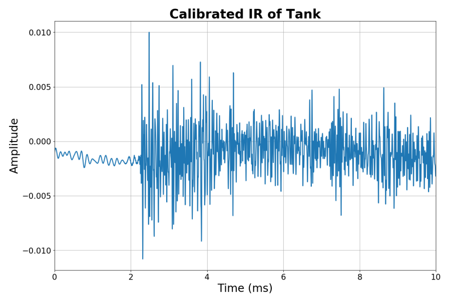
(b)



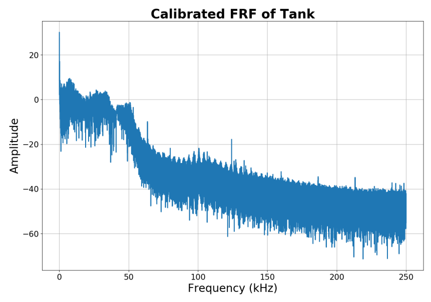
(c)



(d)

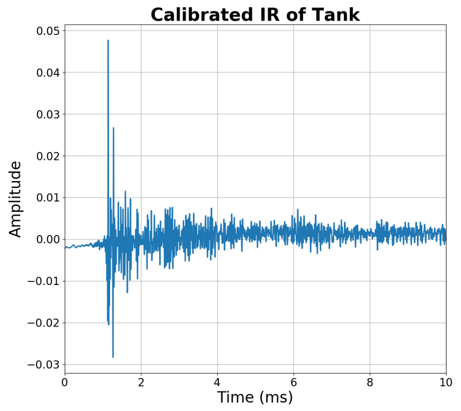


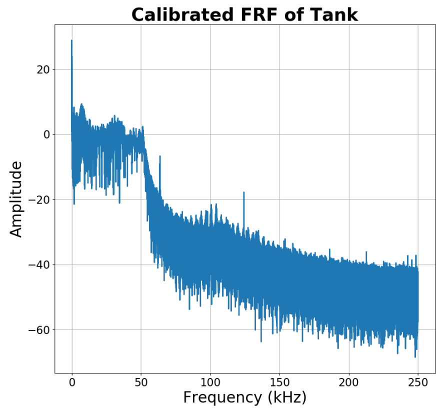
(e)



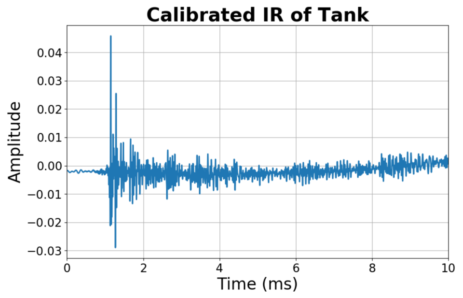
(f)

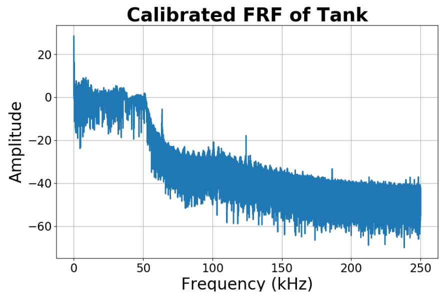
Figure 4.11 Deconvolved impulse and frequency responses for measured 10-50 kHz swept-sine signals at various source and receiver positions along the length of the tank. Fig. 4.11a and 4.11b: source position (0.61 m, 2.83 m, 0.3 m) and receiver position (0.61 m, 1.83 m, 0.3 m). Fig. 4.11c and 4.11d: source position (0.61 m, 2.83 m, 0.3 m) and receiver position (0.61 m, 1.33 m, 0.3 m). Fig. 4.11e and 4.11f: source position (0.61 m, 2.83 m, 0.3 m) and receiver position (0.61 m, 0.83 m, 0.3 m).



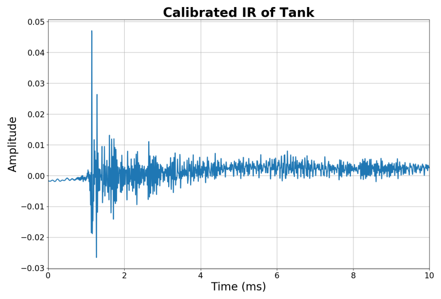


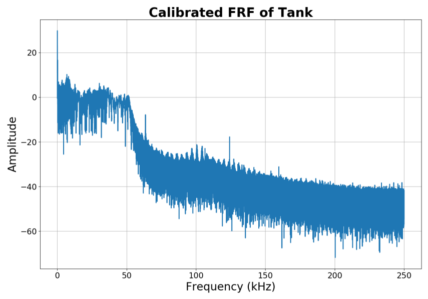
(a) **(b)**





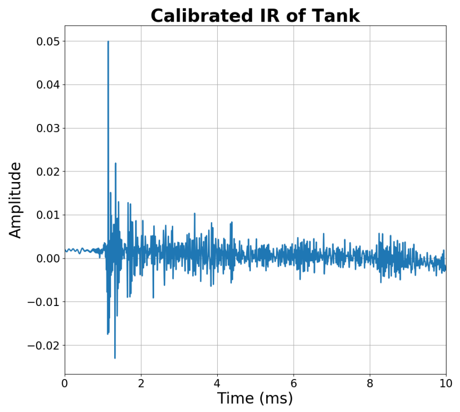
(c) **(d)**



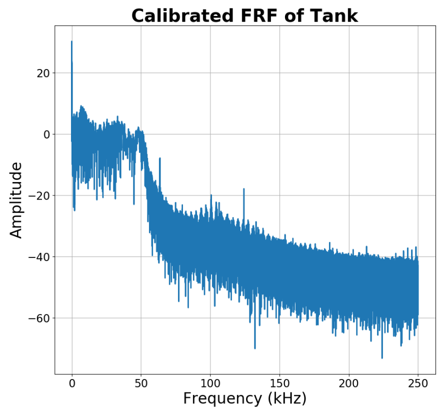


(e) **(f)**

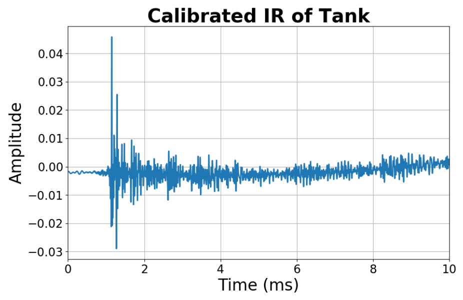
Figure 4.12 Deconvolved impulse and frequency responses for measured 10-50 kHz swept-sine signals at various source and receiver positions along the width of the tank. Fig. 4.12a and 4.12b: source position (0.410 m, 2.130 m, 0.300 m) and receiver position (0.410 m, 1.830 m, 0.300 m). Fig. 4.13c and 4.13d: source position (0.610 m, 2.130 m, 0.300 m) and receiver position (0.610 m, 1.830 m, 0.300 m). Fig. 4.12e and 4.12f: source position (0.810 m, 2.130 m, 0.300 m) and receiver position (0.810 m, 1.830 m, 0.300 m).



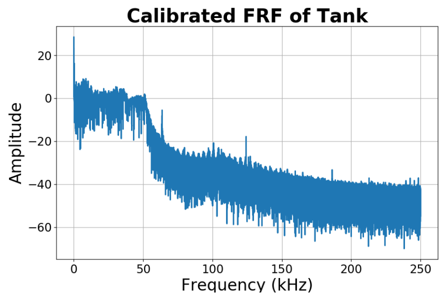
(a)



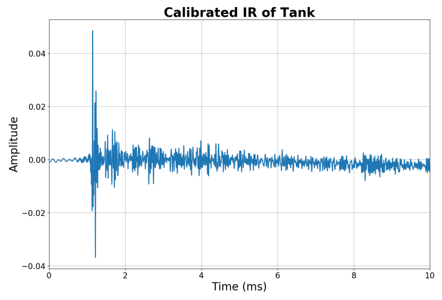
(b)



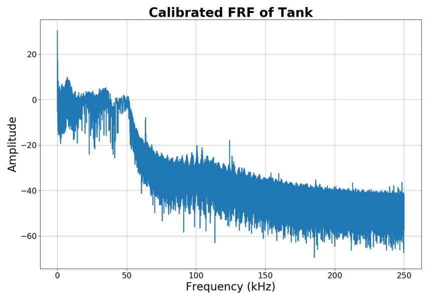
(c)



(d)



(e)

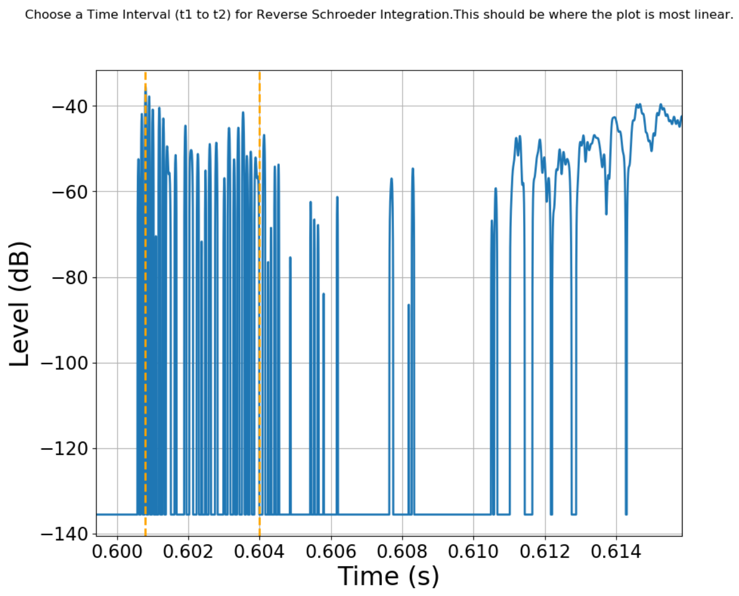


(f)

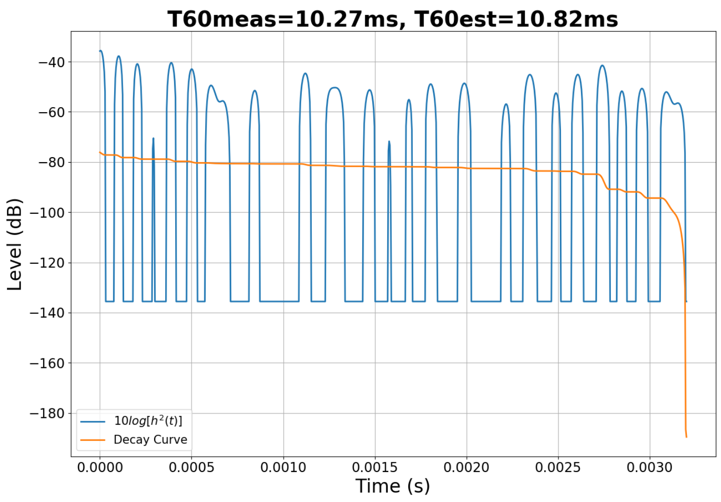
Figure 4.13 Deconvolved impulse and frequency responses for measured 10-50 kHz swept-sine signals at various source and receiver positions along the depth of the tank. Fig. 4.13a and 4.13b: source position (0.610 m, 2.130 m, 0.250 m) and receiver position (0.610 m, 1.830 m, 0.250 m). Fig. 4.13c and 4.13d: source position (0.610 m, 2.130 m, 0.300 m) and receiver position (0.610 m, 1.830 m, 0.300 m). Fig. 4.13e and 4.13f: source position (0.610 m, 2.130 m, 0.350 m) and receiver position (0.610 m, 1.830 m, 0.350 m).

curve of the square of the impulse response through reverse Schroeder integration [49], as discussed in Sec. 3.1.2. As mentioned above, each scan included a total of 729 different source and receiver position pairs to provide more than efficient averaging of measurements. Though these positions are not exactly random, enough positions were obtained to provide more than sufficient data (The standards require 12 random positions.) and to ensure that measurements are not confined within zones of abnormally high or low acoustic energy due to tank modes.

The first step is to define time bounds for the integration: t (time in which the impulse response begins) and t_1(time in which the impulse response decays to the noise floor). These integration bounds for obtaining the decay curve of the impulse response were not measured for all 729 position pairs over all eight cases (four frequency bands, with and without anechoic panels). Instead, time bounds were individually assessed for multiple random position pairs to determine a good estimate of the time bounds, which were then used for all 729 measurements in each frequency band. Integration time bounds of the impulse response commonly vary from measurement to measurement, but they generally trend toward a common value that is sufficient for calculating the decay curve over all 729 measurements for each case. To illustrate the process, the selection of a single measurement's time-bounds are shown in Fig. 4.14. In part (a), the time bounds (dashed lines) are plotted on top of the square of the impulse response on a decibel scale: $10\log_{10}(h^2(t))$. The resultant calculated decay curve can be seen in Fig. 4.14(b). The results of reverse Schroeder integration [49, 50] on the randomly selected position pairs were compared to the estimated value of the reverberation time by monitoring the smootheness of the resultant decay curves. Chosen time bounds are provided in Table 4.1. The reverberation time results from reverse Schroeder integration using these time bounds are discussed in the following sections.



(a)



(b)

Figure 4.14 (a) Magnitude squared of the impulse response, on a decibel scale, with the time bounds indicated by dashed lines. (b). The selected portion from (a) with the calculated decay curve. The general linearity of the decay curve suggests some justification for a diffuse-field assumption that is required by ISO 354.

Table 4.1 Time bounds used for determining the decay curve for each case. Some variation is expected between each measurement and the fact that the values for the measurements with anechoic walls are generally different than those without anechoic walls appears to be coincidental.

	Initial Time Bound t (s)	Final Time Bound t_1 (s)
5-10 kHz Acrylic Walls	0.511	0.5155
10-50 kHz Acrylic Walls	0.511	0.5149
50-100 kHz Acrylic Walls	0.5103	0.514
100-500 kHz Acrylic Walls	0.6004	0.606
5-10 kHz Anechoic Walls	0.6008	0.604
10-50 kHz Anechoic Walls	0.6005	0.6025
50-100 kHz Anechoic Walls	0.6005	0.6031
100-500 kHz Anechoic Walls	0.6005	0.6035

4.2.1 Reverberation Time from Reverse Schroeder Integration

The spatially averaged reverberation times of each of eight cases are obtained via reverse Schroeder integration [49, 50] (Sec. 3.1.2, Appendix B.3.5) and processed through a fractional octave filter (Appendix B.2.2) to obtain 1/30th octave bands. The 1/30th octave band results are seen in Fig. 4.15. Overall values, averaged over the four bandwidths (5-500 kHz) before applying the fractional octave filter, are also provided. The overall reverberation time for the tank environment was originally estimated as 10.82 ms. Using the final time bounds and reverse Schroeder integration, the average reverberation time over the frequency band 5-500 kHz in the acrylic-walled tank is 10.49 ± 2.55 ms and 6.96 ± 2.2 ms for the tank lined with anechoic panels. The anechoic panels reduce the overall reverberation time of the tank by over 33% and work well over the advertised bandwidth (20-200 kHz). The reduction in reverberation time corresponds with a general increase in absorption within the tank as expected, and in practice reduces the settling time required between measurements.

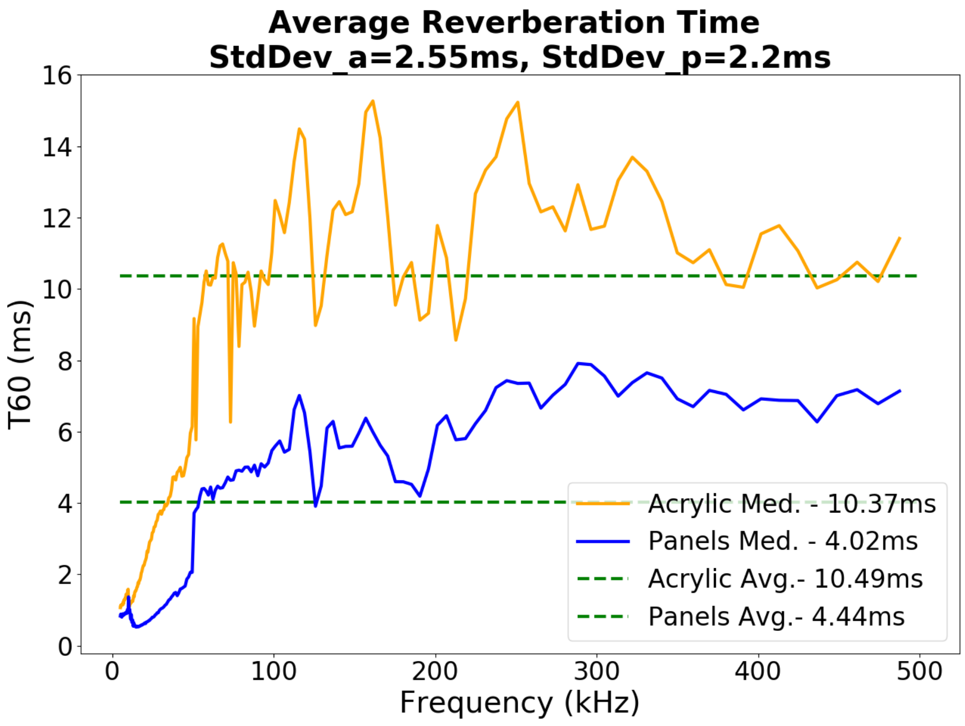


Figure 4.15 Frequency-dependent, spatially averaged reverberation time from reverse Schroeder integration at 1/30th fractional octave bands from swept-sine signals within a tank with (orange) and without (blue) anechoic panels on the side walls. These results come from averaging over all 729 source-receiver pair measurements within the tank. The overall reverberation time for the tank was estimated to be $T_{60est} = 10.82$ ms which can be compared to the median values shown in the legend. The standard deviations for the acrylic (a) and anechoic panel (p) cases are listed in the title.

The degree to which this reduction in reverberation time improves the tank's ability to perform as a scale-model ocean depends on the measured panel absorption. Knowing the reverberation time, the modified Norris-Eyring equation (Eq. 3.2) may be solved to obtain the spatially averaged absorption of the tank boundaries (Sec. 3.1.4).

4.2.2 Approximate Spatially Averaged Absorption

The spatially averaged absorption coefficients with and without the anechoic panels are obtained from the frequency-dependent reverberation time by solving the modified Norris-Eyring equation (Eq.3.2) using the code in Appendix B.3.6. (The absorption of the anechoic panels could alternatively be explored using the algorithm in Appendix B.3.7 to compare reverberation times across measurements with added material. The results of the 1/30th octave band spatially averaged absorption with no panels in the tank and with the assumption of zero propagation absorption are presented in Fig. 4.16. The standard deviation across the 729 measurement locations are shown as vertical bars. The same statistical analysis of the measurement with the anechoic panels and is shown in Fig. 4.17. The overall absorption with the anechoic panels (0.6089) is greater than without (0.4708) as expected: When averaged over the entire bandwidth from 5-500 kHz, the average value is 0.77 with the panels compared to 0.47 without the panels. The absorption of the panels is not large enough of a value to easily assume the tank may be modeled as a near open ocean environment without (or with minimal or negligible) side wall reflections. This is because a truly anechoic environment would assume a wall absorption of 0.99. However, this condition may perform reasonably as an open ocean as discussed in Sec. 4.3.

The spatially averaged wall absorption curves in Figs. 4.16 and 4.17 were obtained using the assumption that the water-surface interface is a pressure-release boundary. When accounting for the small absorption (or small finite-impedance) of the water-air surface instead of assuming a perfectly pressure release surface, the spatially averaged wall absorption shown as dashed lines in Fig. 4.18 are obtained and do not differ from the original ones. Thus, this small shift in the air-surface boundary condition appears to be negligible, especially when no anechoic panels are in place. This agreement is expected and shows the viability of using the assumption of a pressure-release water-air surface.

The overall increase in absorption with the panels suggests that the addition of the anechoic panels along the side walls improved the absorption of the walls of the tank significantly in the desired ultrasonic bandwidths.

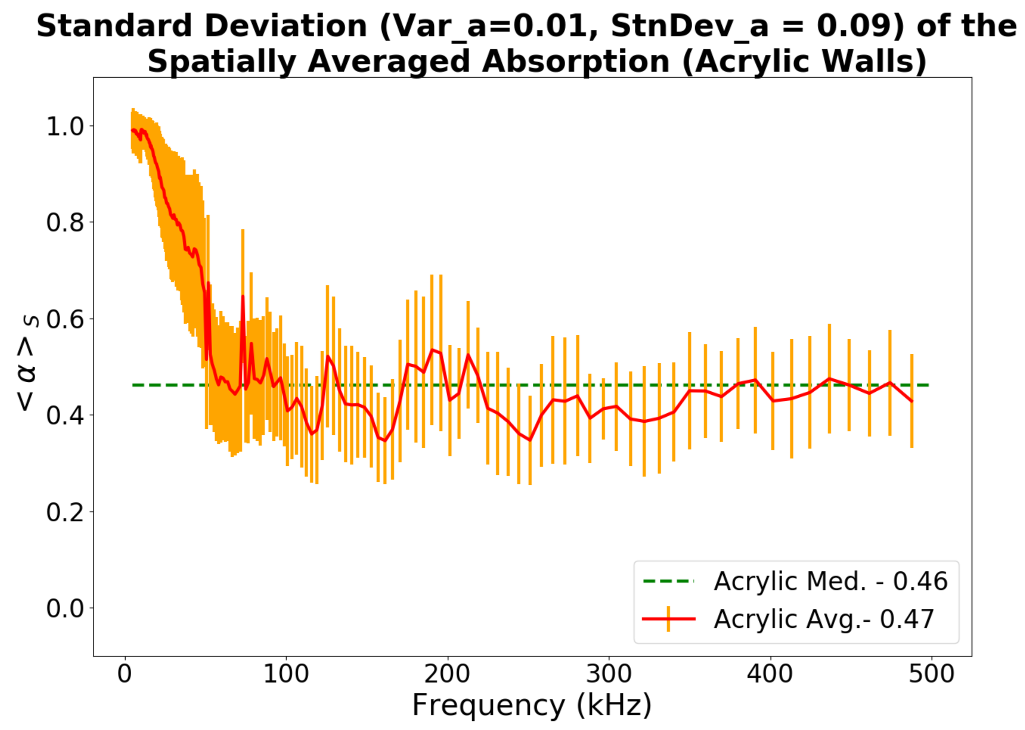


Figure 4.16 Statistical analysis for the frequency-dependent spatially averaged absorption coefficient calculated from the reverberation time for the tank without the anechoic panels averaged over 729 measurements. The standard deviation is plotted suggesting reasonable uncertainty for the calculated absorption coefficient. The median value over frequency is plotted as a green dashed line and listed in the legend. The average overall absorption over the full bandwidth is also shown in the legend.

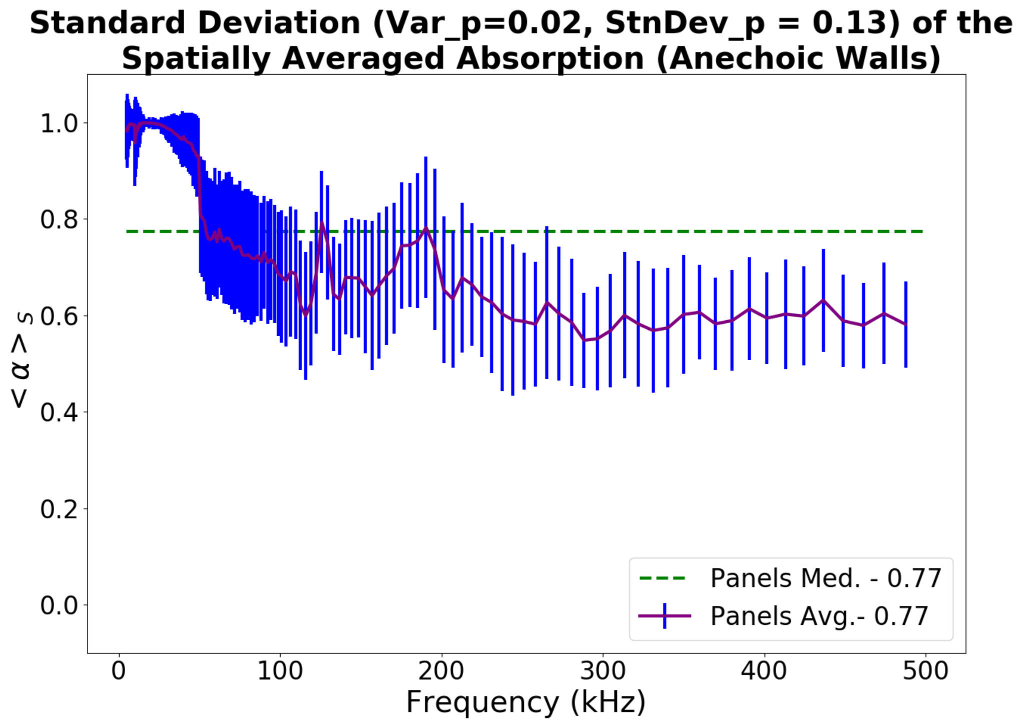


Figure 4.17 Statistical analysis for the frequency-dependent spatially averaged absorption coefficient calculated from the reverberation time in the tank with the anechoic panels averaged over 729 measurements. The standard deviation is plotted suggesting reasonable uncertainty for the calculated absorption coefficient. The median value over frequency is plotted as a green dashed line and listed in the legend. The average overall absorption over the full bandwidth is also shown in the legend.

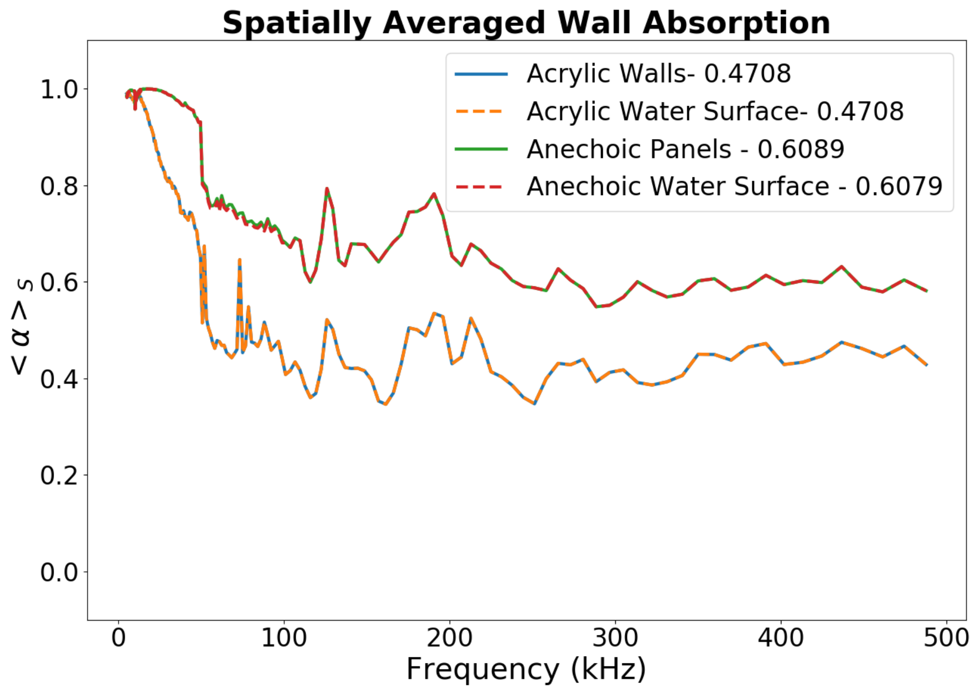


Figure 4.18 Frequency-dependent spatially averaged absorption coefficient from a 1/30th fractional octave band analysis of swept-sine signals within a tank with and without the anechoic panels. The solid lines are from calculation that assumes a pressure-release boundary condition at the water-air interface. The dashed lines show the absorption obtained when the small finite-impedance of the water-air surface is included in the calculations. These calculations assume no propagation absorption losses. The single value overall absorption values in the legend represent the average absorption for the full bandwidth before applying the 1/30th fractional octave band filter.

The frequency-dependent, spatially averaged, wall absorption $< \alpha >_S$ calculated while accounting for propagation absorption effects (i.e., while applying Eq. 3.3) is presented in Fig. 4.19. In general, the absorption is reduced when a portion of the overall measured absorption is accounted for by propagation losses as seen particularly below 50 kHz. The overall absorption of the acrylic walls accounting for propagation absorption appears to have unexpectedly increased as seen from the sudden spikes from 50-100 kHz. Above 135 kHz the calculated absorption of the walls accounting for the effects of propagation losses drop well below zero unexpectedly. This suggests significantly higher propagation absorption than wall absorption with increased frequency. However, $< \alpha >_S$ obtained using Eq. 3.37 and shown in Fig. 4.19 follow an unexpected trend, particularly above 135 kHz where the value drops to large negative values, it is hypothesized that potential exists for the Ainslie and McColm [45] model to have an upper limit of precision or limitations for small-scale measurements within tanks. This potential limit makes sense since the anechoic panels are advertised originally as having good absorption from 20-200 kHz and the model was originally developed and evaluated on open ocean propagation. These improbable results coupled with low variance over 729 measurements suggest a need to further investigate other propagation models.

The spatially averaged overall absorption coefficient averaged for the four bandwidth measurements from 5-500 kHz are compared to that obtained using the published impedance of acrylic in Table 4.2. The value of specific acoustic impedance of clear acrylic of 3.26×10^6 Pa s/m was found on an open-source nondestructive testing webpage (https://www.ndt.net/links/proper.htm). Using the accepted specific acoustic impedance of water ($z_w = 1.5 \times 10^6$ Pa s/m) and of air ($z_a = 415$ Pa s/m), the absorption coefficient of clear acrylic was estimated as 0.6302. The overall absorption (not frequency-dependent over 5-500 kHz) coefficient for one-inch thick clear acrylic in the tank was found to be 0.4708 without or 0.4937 with including propagation losses, and the overall absorption coefficient of the acrylic walls lined with anechoic panels was determined to be 0.609 or 0.5789 including propagation losses. The estimated overall values are similar to the spatially averaged wall

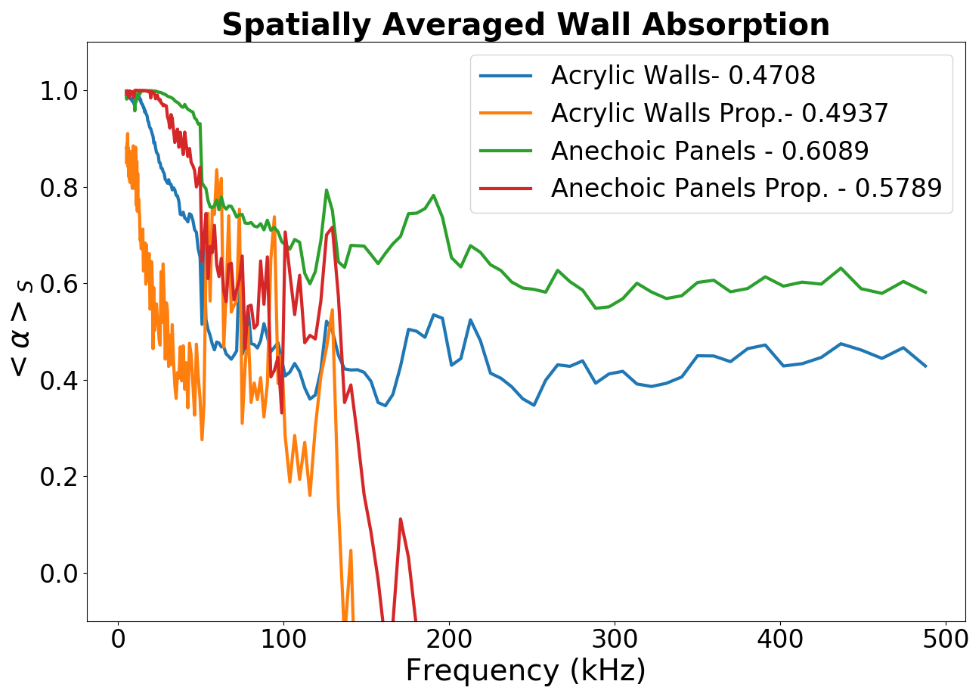


Figure 4.19 Frequency-dependent spatially averaged absorption coefficient from a 1/30th fractional octave band analysis of swept-sine signals accounting for propagation absorption through the water within a tank with (red) and without (orange) the anechoic panels on the side walls. It is hypothesized that there may be an upper limit of precision for the Ainslie and McColm model above 135 kHz where accounting for the propagation absorption has caused both the acrylic and anechoic wall absorption to drop well below zero, which should not be true particularly with such small propagation distances.

absorption values over the 5-500 kHz band in Fig. 4.18. The spatially averaged absorption coefficients, $< \alpha >_S$, for the acrylic and anechoic boundaries are necessary parameters for developing an acoustic propagation model with finite-impedance boundaries.

4.3 Applied Finite-impedance Boundary Model

The spatially averaged absorption coefficients are used for the finite-impedance boundary model in Eq. 3.34 (see Appendix B.4.3). As discussed in Sec. 3.3.3, $< \alpha >_S$ is used for defining the boundary conditions. An example of the resulting modeled transmission loss is displayed in Fig. 4.20 for a measurement without panels over the 10-50 kHz band. The measured and estimated absorption values from Table 4.2 are used in the model to obtained the two modeled frequency response or transmission loss from 10-50 kHz. The great agreement provides confirmation that calculation of the spatially average absorption coefficient using the TTS response is consistent.

The finite-impedance model with Robin boundary conditions provides a more accurate model than the idealized tank rigid-wall (Neumann) boundary with a pressure-release (Dirichlet) surface model (Eq. 3.28), as illustrated in Fig. 4.21 for the tank with acrylic walls (a) as well as with anechoic panels lining the walls (b). The Pierce finite-impedance boundary model improves agreement with measurements compared to the idealized rigid-boundary model which is essentially noise at this bandwidth. Decent agreement of the Pierce model is also seen with the ORCA model for both cases. The vertical offset of the models from the measured data occurs because the source strength is not accounted for in these models and an amplitude of 1 (corresponding to a point source) is assumed for each model, which should account for the 20 dB offset. This evaluation constitutes an initial comparison of models to the measured data. An extensive evaluation of or development of the models is beyond the scope of this thesis.

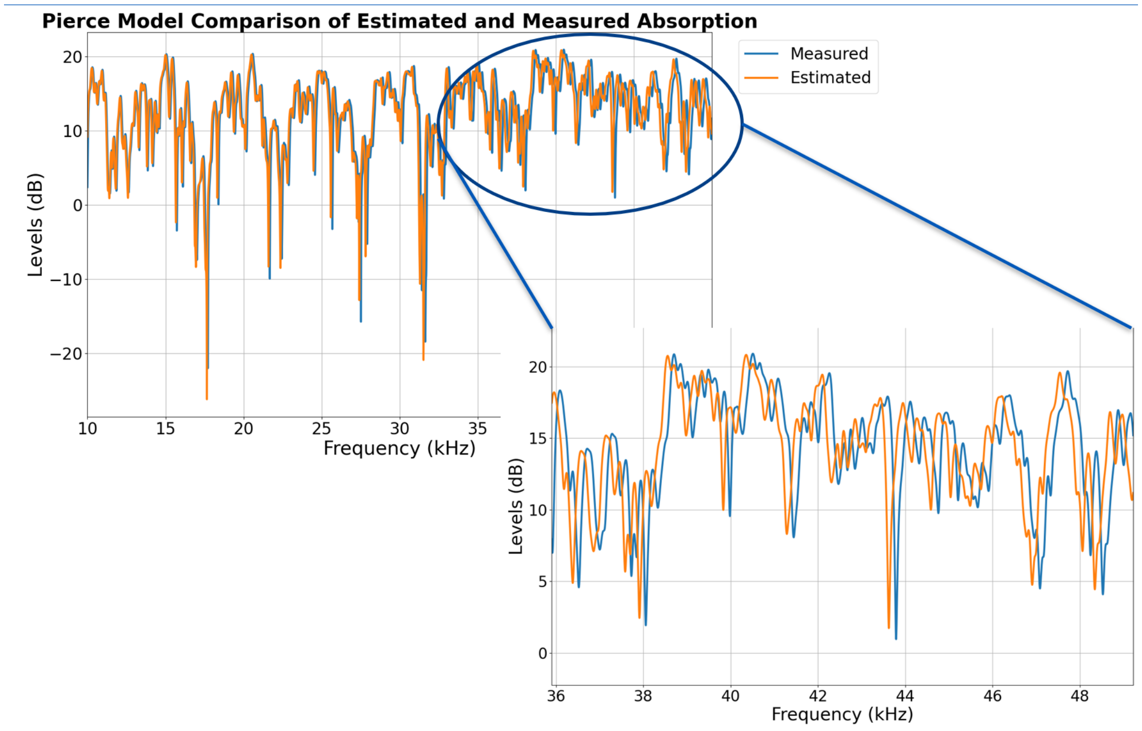
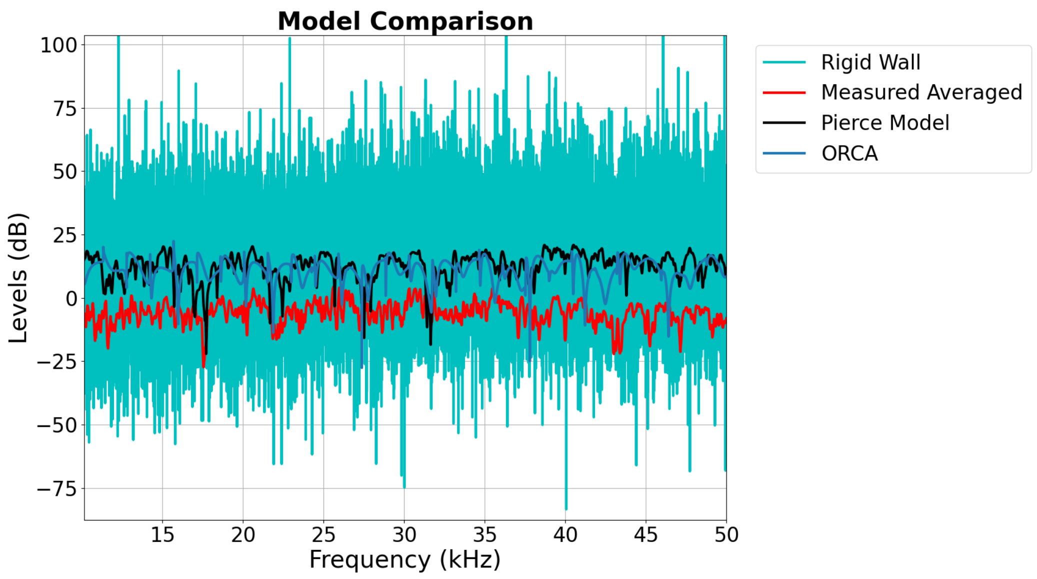
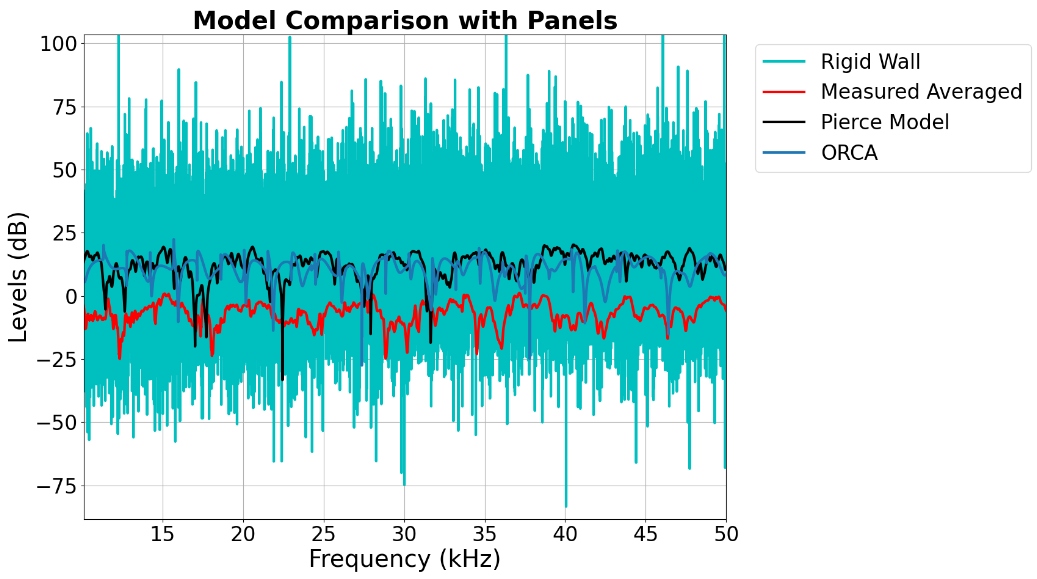


Figure 4.20 Transmission loss modeled with the Pierce finite-impedance model using the spatially averaged absorption coefficient by fractional octave bands from swept-sine signals within a tank with anechoic panels applied to the side walls and one obtained from published values of the impedance of acrylic. For this case, the source was at (0.41 m, 2.13 m, 0.25 m) and the receiver at (0.41 m, 1.83 m, 0.25 m). The frequency band from 36-50 kHz has been zoomed in for comparison.

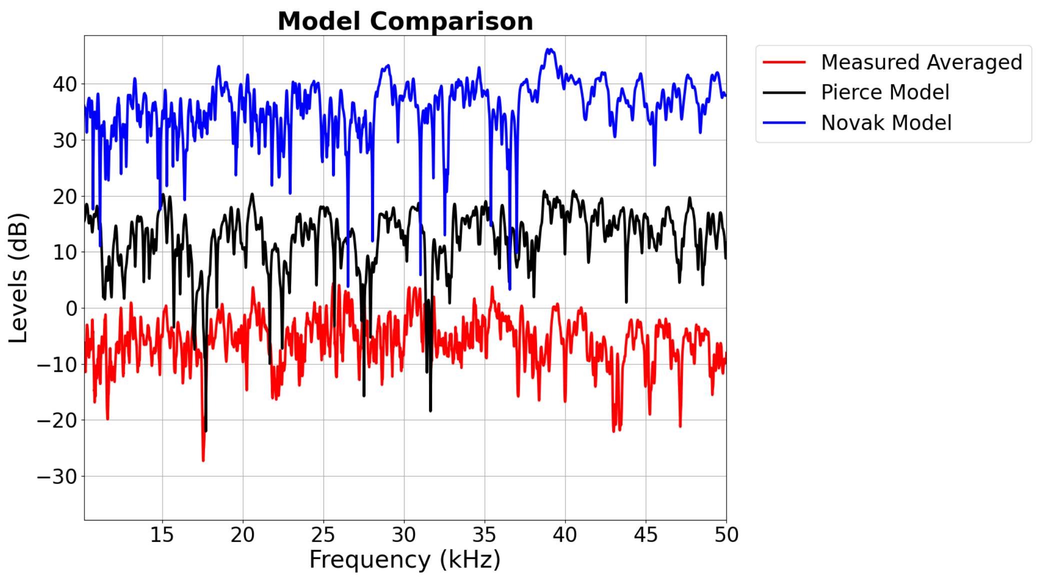


(a) Acrylic-walled tank.

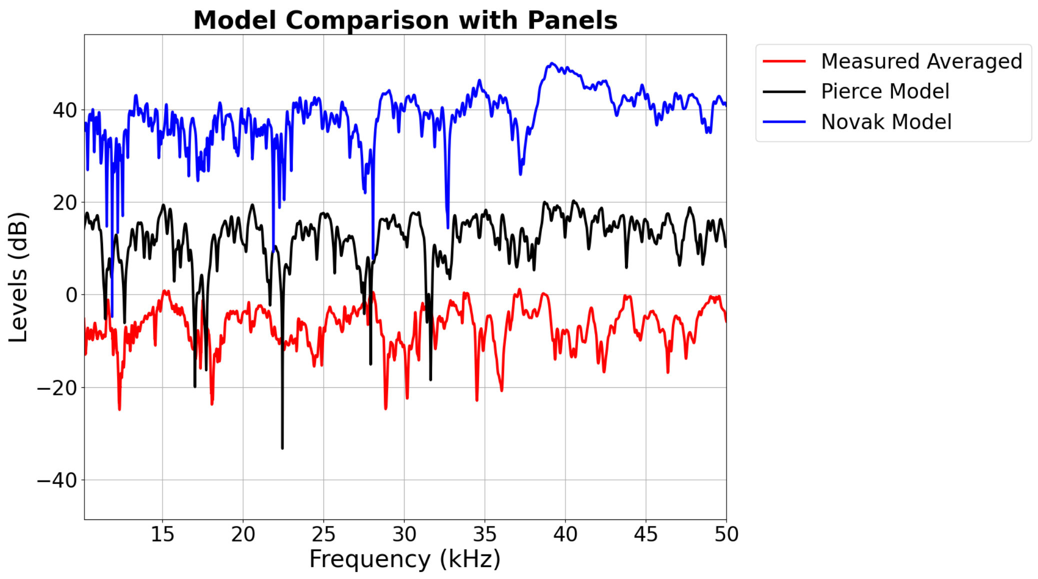


(b) Anechoic-lined tank.

Figure 4.21 Comparison of transmission loss for the idealized rigid-wall model (Sec. 3.3.2) with the Pierce finite-impedance boundary model, ORCA model, and measured values. A source strength of 1 is assumed for these models because this was only an initial comparison which should account for the approximately 20 dB difference overall. Models have the measured absorption applied.

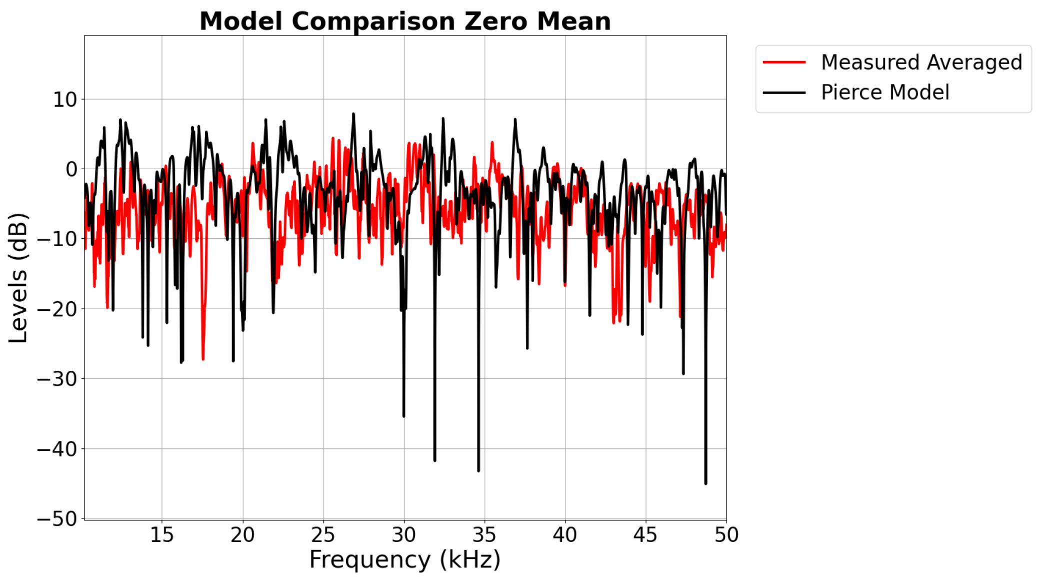


(a) Acrylic-walled tank.

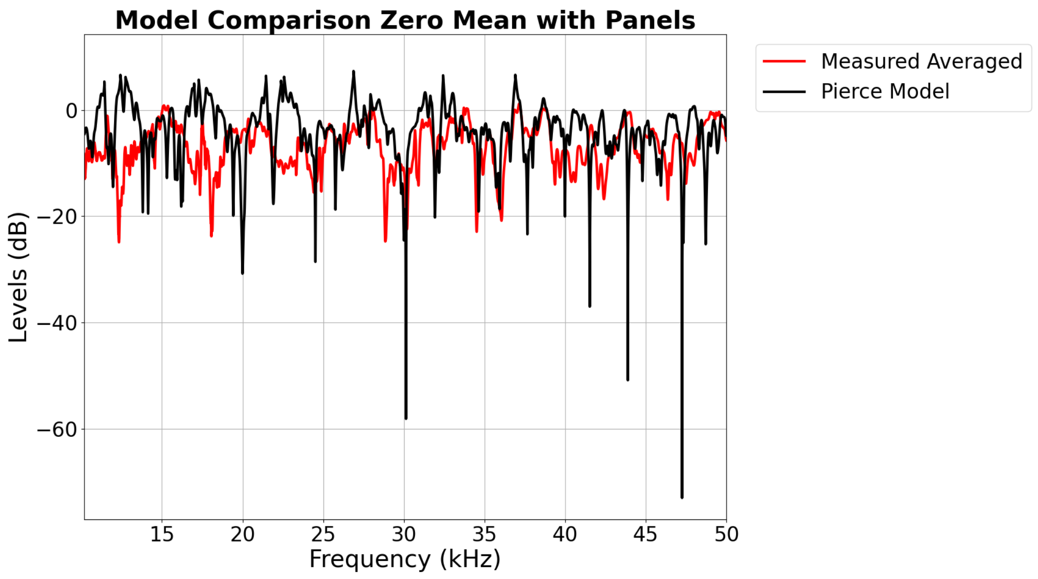


(b) Anechoic-lined tank.

Figure 4.22 Comparison of transmission loss from the Novak finite-impedance boundary model (Sec. 3.3.2) with the Pierce finite-impedance boundary model and measured values. A source strength of 1 is assumed for these models because this was only an initial comparison which should account for the approximately 20 dB difference overall. Models have the measured absorption applied.

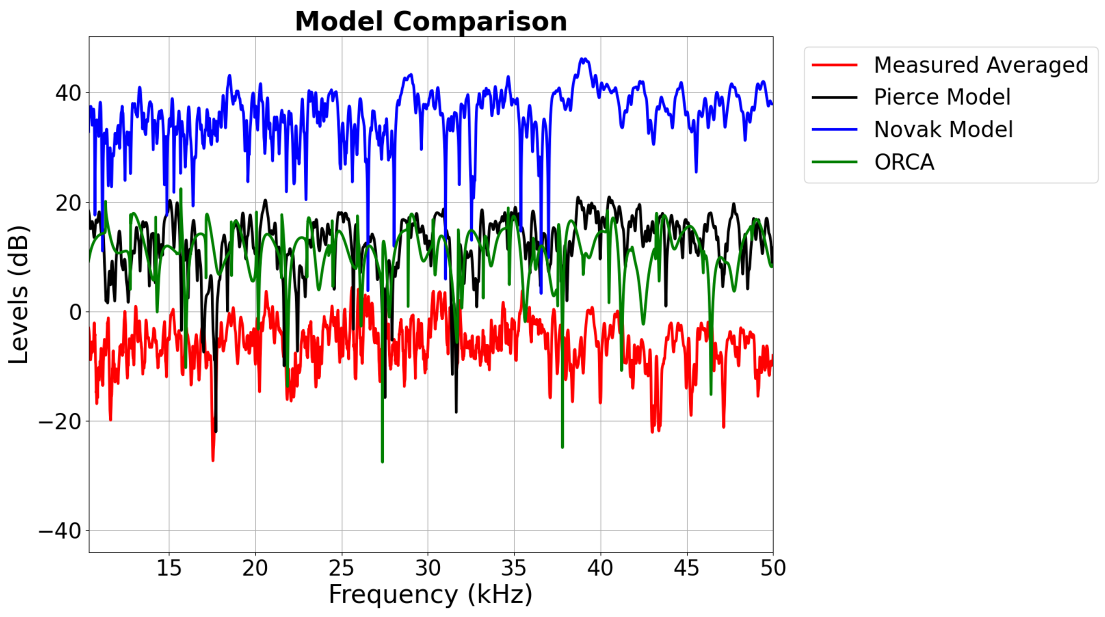


(a) Acrylic-lined tank.

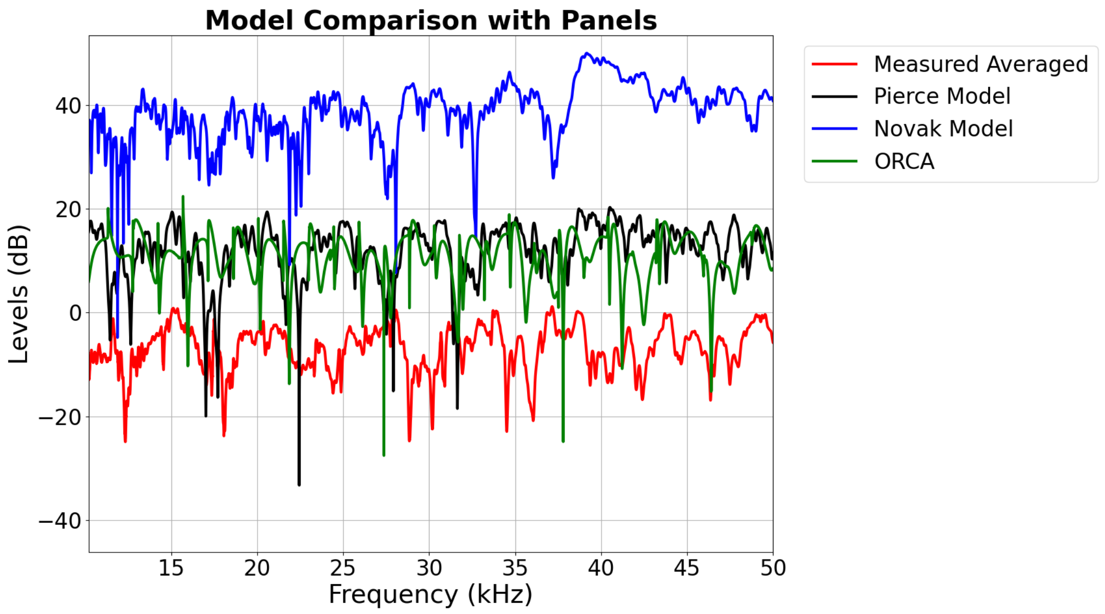


(b) Anechoic-lined tank.

Figure 4.23 Comparison of the transmission loss of the Pierce finite-impedance boundary model with measured values without (a) and with (b) anechoic panels lining the tank walls. A zero mean scaling was applied across frequency for ease of comparison. It can be seen that the Pierce finite-impedance boundary model shows some frequency-dependent agreement with measured results.



(a) Acrylic Walled Tank



(b) Anechoic Walled Tank

Figure 4.24 Comparison of transmission loss from ORCA, the Pierce finite-impedance boundary model and measured values without (a) and with (b) anechoic panels lining the tank walls.

Initial evaluation of the frequency response provided by impedance boundary tank models using *in situ* calibrated impedance boundary characterization is seen in Fig. 4.22 for the acrylic tank (a) as well as with the anechoic panels lining the tank walls (b). The offset is due to an assumed model amplitude of 1 and allows for a better visual comparison of the Pierce and Novak models with measured data. There is not clear agreement between the models and the measured data with only acrylic walls. When the anechoic panels are applied to the walls and the models, greater frequency-dependent agreement in the general trends of the modeled and measured transmission losses are seen particularly below 30 kHz.

To better portray this trend, a zero mean normalization is applied to the modeled and measured values. The results in Fig. 4.23 show some agreement between Pierce's [43] model and measured values especially from 25-40 kHz with anechoic panels. It is shown that the anechoic panels appear to improve the ability of the Pierce model to represent actual tank measurements when measured boundary impedance is applied. The potential of using the Pierce model and measurement-based absorption coefficients to model sound propagation in the tank is compared to the modeled values for ORCA [15] as a benchmark solution in Fig. 4.24. (This example does not have the zero-mean applied.) The finite-impedance model (Pierce model) shows reasonable initial agreement with the overall amplitude of the ORCA model with some similarities in shape particularly from 25-40 kHz as seen in Figs. 4.24a and 4.24b. The Pierce finite-impedance model demonstrates initial potential for being a good intermediary model between tank measurements and an open ocean model. These data also demonstrate the ability of the anechoic panels in improving the models and measurements for open ocean modeling within a tank environment as seen with improved model agreement with anechoic panels applied to the walls seen in Fig. 4.24b. Future model-data comparisons are needed to fine-tune the modeling of sound propagation in the tank.

4.4 General Tank Characterization

Determining the boundary characterization from *in situ* calibrated tank responses provides additional characterization of acoustic parameters within the tank. The most prominent parameters obtained alongside the measured boundary absorption are discussed below.

4.4.1 Schroeder Frequency

The Schroeder frequency limit was estimated in Sec. 3.1.2, following Eq. 3.1, to range between 1000 Hz and 3000 Hz depending on water depth, temperature, and salinity. This limit determines cutoff frequency for effective modal modeling of acoustic propagation within the tank. The estimated Schroeder frequency for the parameters (water depth of 0.5 m and temperature of 20.88° C) of the tank during data collection for this thesis was $f_{S\text{est}} = 1311$ Hz. Through measuring the reverberation time of the tank (Sec. 4.2.1), it was independently determined that the Schroeder frequency for the acrylic-walled tank was 2618 Hz. The Schroeder frequency limit shifted to 2416 Hz (overall) when the anechoic panels were applied to the walls of the tank. These values confirm that the dimensions as well as the absorption of the tank make it qualified for the intended ultrasonic bandwidth. The Schroeder frequency can also inform assumptions when operating with bandwidths below this limit. The Schroeder frequency limit is also dependent on the speed of sound, which estimate was confirmed by measurements.

4.4.2 Speed of Sound

Several equations have been proposed for calculating the speed of sound in the ocean. Since the tank can have a maximum depth of 0.92 m and is currently filled with tap water, the primary source of sound speed variability within the tank is the temperature of the water. The results of Garrett's formulation as a function of temperature are compared to two other classical models: the Medwin

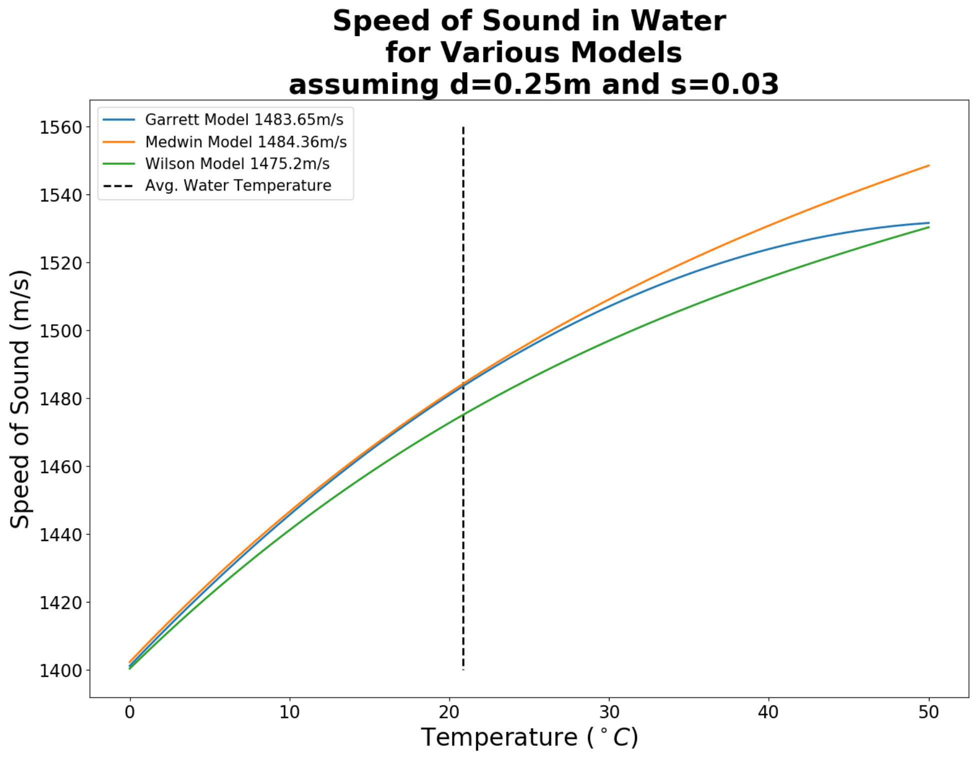


Figure 4.25 Speed of sound in water compared using three classic models. The average tank water temperature for data collected in this thesis is shown as the dashed line, and the speed of sound predicted by each model for the average water temperature in the tank is given in the legend. These values are determined with the assumption of low water depth (d) and low salinity (s), which are common for the tank environment.

model [54] and the Wilson model [55]. The effect of temperature on the sound speed in the water according to various models is illustrated in Fig. 4.25. According to Garrett's formulation [3] (Eq (11.26)), the speed of sound in water for a depth of 0.25 m (average depth in a tank with 0.5 m of water) and a low salinity of 0.03 ppt at a temperature of 20.88°C is 1483.65 m/s, which is close to the Medwin value and about 10 m/s higher than the Wilson value.

To further justify this sound speed approximation from Garrett's model, measurements were taken to determine the acoustic travel time at different distances. The measurements were compared using a cross-correlation function to determine a time-delay vs range. The corresponding sound speeds are shown as a function of range in Fig. 4.26. Except for a few outliers, the sound speeds

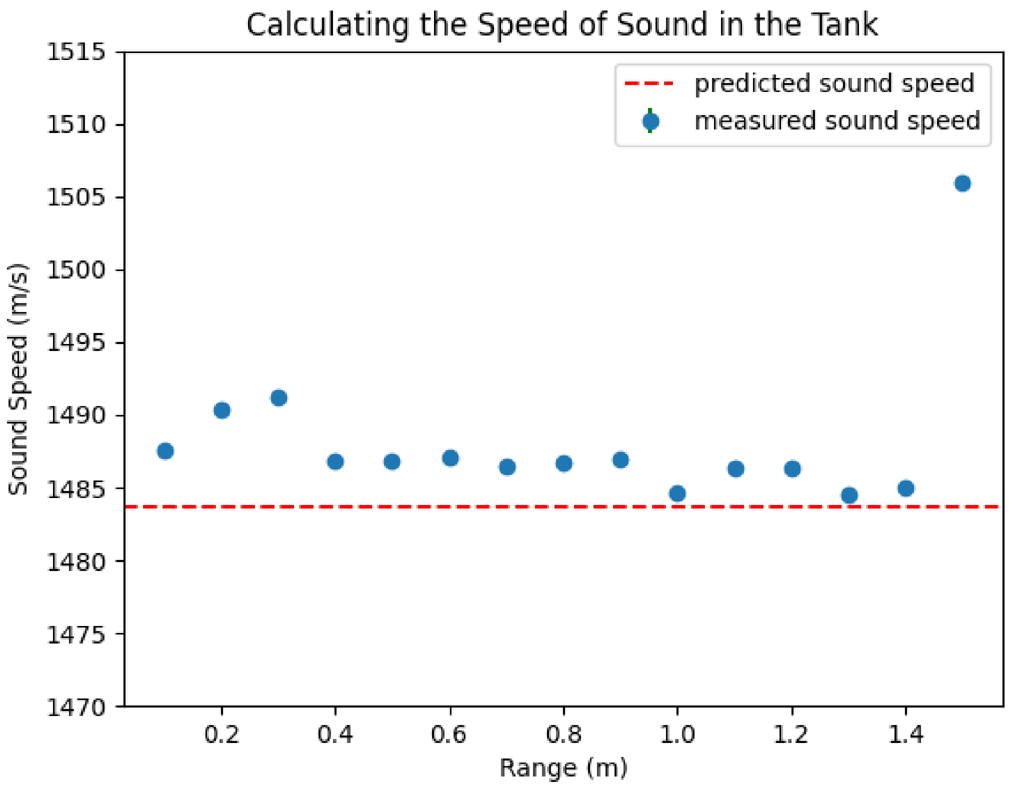


Figure 4.26 Speed of sound in water measured in the tank through correlation of range-dependent scans. A comparison is made to the sound speed estimated by Garrett's model for an average tank water temperature of 20.88°C with low water depth and an assumed low salinity. The measured data has an average of 1487 m/s compared to the model estimate of 1484 m/s from Garrett's model.

from the cross-correlation approach are within 3 m/s of the value obtained from Garrett's model. This comparison indicates that the estimated value based on the Garrett model may be used in the future to adjust for variations in water temperature and salinity when processing data.

4.4.3 Tank Noise

Another important characteristic of the tank that needs to be understood is the ambient noise present in recordings. This ambient noise profile is of interest, even though the deconvolution method

Table 4.2 Summary of estimated and measured data for the overall bandwidth from 5-500 kHz. This table provides data for the acoustic characteristics (absorption coefficient, reverberation time, Schroeder frequency limit, speed of sound in water) of the tank with acrylic and anechoic boundaries.

	Estimation Parameters	Estimated Acrylic	Measured Acrylic	Measured Anechoic

Absorption Coefficient	$z_{ac.} = 3.26e6\,\frac{Pa\,s}{m}$
	$z_{w.} = 1.5e6\,\frac{Pa\,s}{m}$
	$z_{air} = 415\,\frac{Pa\,s}{m}$
		0.6302	0.4708
			z=4.872E6
			
			0.4937 (prop)
			z=4.576E6
				0.609 (backed)
				z=3.426E6
				
				0.5789 (w/prop)
				z=3.682E6
				
				0.8629 (panels only)
				z=1.977E6

Reverberation Time	Rigid Walls	10.82 ms	10.49 ms	6.96 ms
Schroeder Frequency Limit	**c = 1483.65 m/s**	**2387.54 Hz**	**2618.31 Hz**	**2416.37 Hz**
Speed of Sound	**T = 20.88℃** **d = 0.5 m** **S = 0.03 ppt**	**1483.65 m/s**	**1486.92 m/s**	**1486.92 m/s**

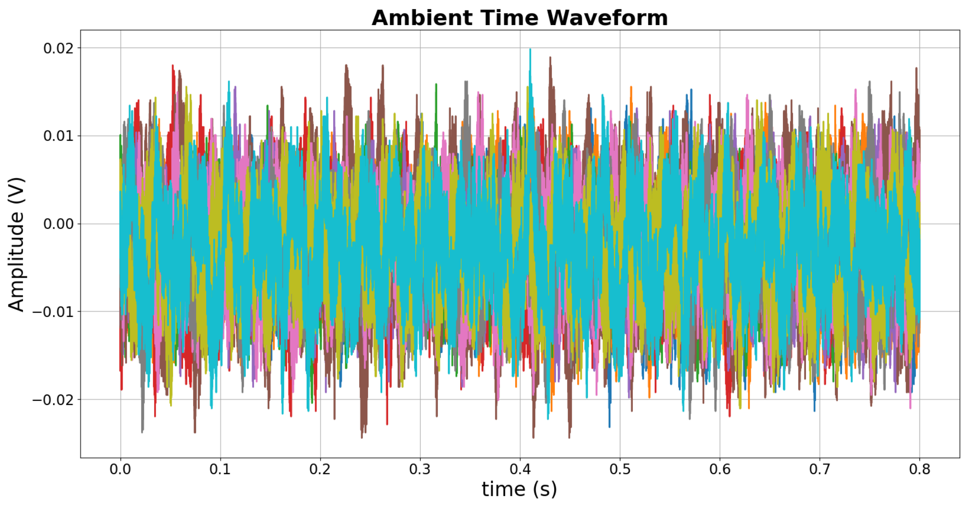


Figure 4.27 Raw time waveforms of the ambient noise in the tank. Ten separate ambient measurements are plotted together.

(Sec. 3.2.1) handles noise well. A set of ambient noise measurements were taken on the 28th of June 2021 as described in Appendix A.5. The resulting signals for ten measurements are displayed in Fig. 4.27. These output voltages (which can be changed to acoustic pressure when a voltage sensitivity is applied) are normally distributed over all ten measurements. The distribution for one of the measurements is shown in Fig. 4.28. The ambient noise in the tank environment is most prominent below the Schroeder frequency (approximately 2 kHz) as seen in Fig. 4.29. All ten measurements appear to be fairly consistent in both frequency content as well as amplitude. This consistency indicates that the deconvolution method (Eq.3.18) using the TTS response should account for uncorrelated ambient noise within the tank without the need for multi-scan averaging as discussed in Sec. 3.2.

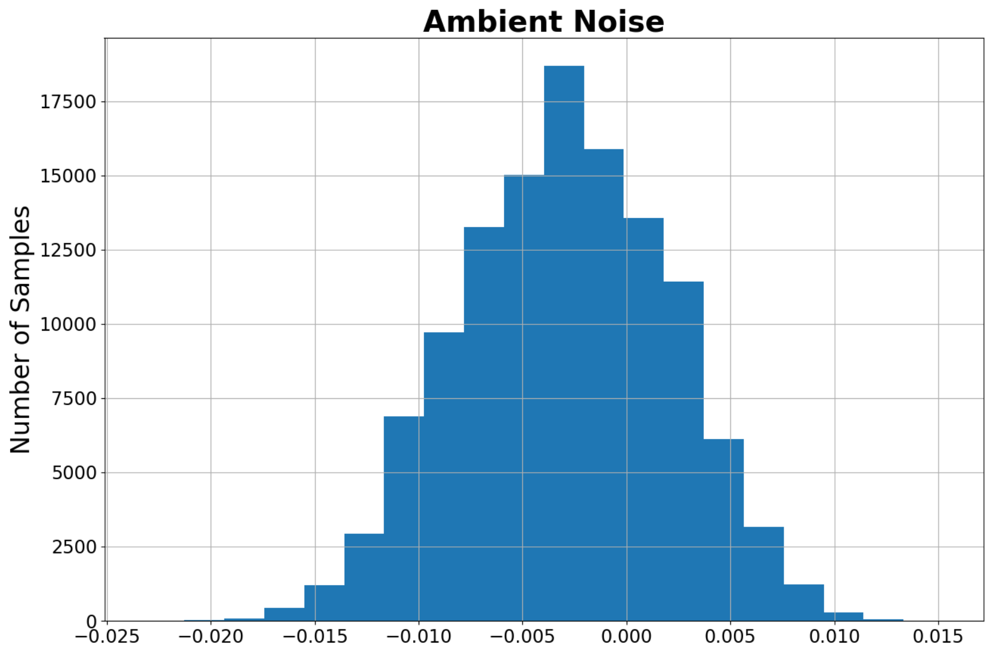


Figure 4.28 A histogram of the output voltages from one of the ambient measurements reveals a normal distribution about the mean. Nine other measurements were analyzed and shown similar normal distributions. Note the DC offset of approximately 3 mV.

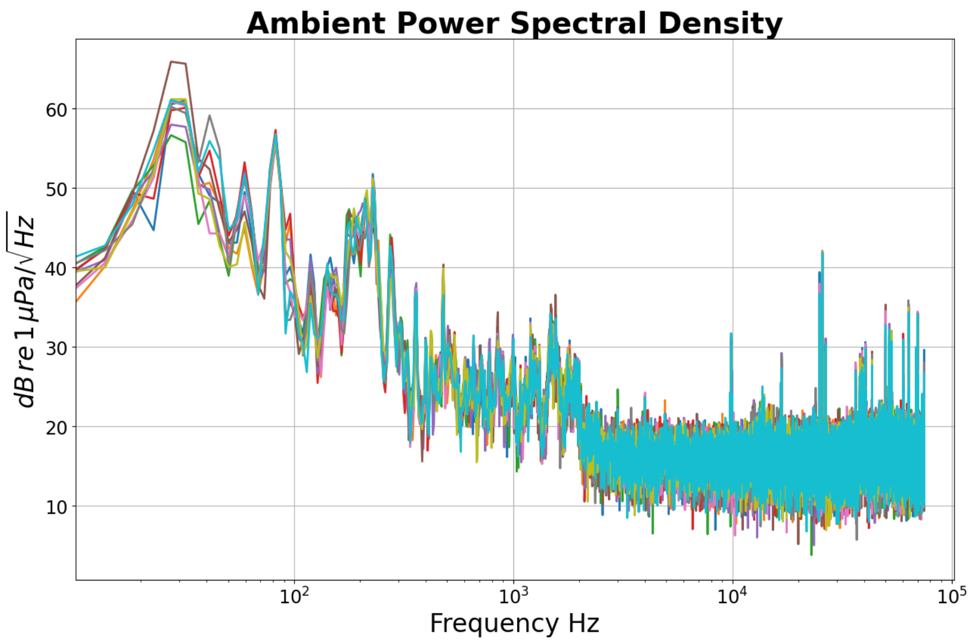


Figure 4.29 Power spectral density of the ambient noise measurements showing that the majority of the ambient tank noise is below 2 kHz (below the Schroeder Frequency). Ten separate ambient measurements are plotted together. The high frequency peaks are hypothesized to be electromagnetic radiation from ambient sources such as the robotic arms.

Chapter 5

Conclusion

The BYU underwater acoustics research laboratory has been designed and built from the ground up as part of this thesis work with both a high level of visual aesthetic as well as functionality. The research table was even custom designed and built to both match the aesthetic of the tank as well as offer many benefits for researchers while taking measurements such as monitoring equipment and maintaining effective cable management. The acoustic nature of the water tank environment along with carefully selected anechoic panels have been evaluated and a measurement chain has been developed and validated to provide quality acoustic measurements in a safe and user friendly way. The measurement chain uses a robotic positioning system controlled by custom software (ESAU) developed to precisely place transducers within the tank for measurements as well as generate signals and collect data. Each piece of equipment has been selected to handle a wide variety of future applications and developed to be safe and user friendly so future students may learn to perform valuable research with ease. This functionality includes a significant amount of automation and remote capabilities.

Beyond the development of a full measurement chain with many capabilities, procedures have been developed and documented to care for the equipment and tank in the lab. These procedures include the active lab documentation in Appendix C which will continue to be updated by future students to address the growing needs of the lab. Along with procedures for caring for the lab, tools have been developed to care for the equipment. The filtration system especially was developed in order to address the concerns of maintaining consistent control of water environmental characteristics and avoid bubbles which may significantly impact measurements. The laboratory has been developed and proven functional and reliable for student research purposes in underwater acoustics.

The BYU underwater acoustics tank and measurement chain have proven their ability to allow for effective underwater acoustic measurements particularly above the Schroeder frequency limit. The acoustic characterization of the tank environment illustrates how acoustic energy propagates and behaves within the tank environment over fractional octave bands within the bandwidth 5-500 kHz. This characterization provides an opportunity to study scaled acoustic measurements in a controlled laboratory environment and for developing a good frequency-dependent acoustic propagation model for the tank.

An *in situ* calibration method, as discussed in Sec. 3.2.2, has been developed, which allows for a systematic accounting for the frequency dependence of all transducers within the measurement chain. The frequency dependence of the measurement chain, referred to as the through-the-sensor (TTS) response, can then be removed from measured data via deconvolution. The deconvolution method (Eq. 3.18) also helps to provide an accounting for uncorrelated ambient noise within the tank without the need for multi-scan averaging. The ambient noise was shown to be most prominent below the Schroeder frequency limit, which is well below the bandwidth of interest considered in this thesis. The TTS response, or calibration, applied to subsequent measured data through deconvolution gives a more precise response of the tank environment.

This more precise response of the tank on sound propagation, especially the impulse response $h(t)$, is required for effectively characterizing the acoustic boundary conditions of the tank. The impulse response is used to determine the reverberation time T_{60} of the tank by determining the decay curve of $h(t)^2$ through reverse Schroeder integration. The resultant reverberation time as well as the measured Schroeder frequency (which is proportional to the T_{60} for the acrylic-walled tank) agreed well with the predicted values. By fractional-octave band filtering, the resultant frequency-dependent reverberation time, $T_{60}(f)$, was obtained for frequencies 5-500kHz in 1/30th octave bands.

The frequency-dependent reverberation time $T_{60}(f)$ was used to estimate the frequency-dependent spatially averaged acoustic absorption coefficient $< \alpha >_S (f)$ by solving the modified Norris-Eyring equation (Eq. 3.2) for both the acrylic-walled tank as well as the tank with anechoic panel lined walls. This characterization helped to better quantify the particular boundary conditions of the tank and provided a clear understanding of the benefit provided by the anechoic panels. This gives a more precise absorption characterization over a larger bandwidth than provided by Precision Acoustics; their information was obtained for panels with different backing (unbacked or backed by a 5mm thick steel plate). Furthermore, the absorption characterization of the tank boundaries confirmed the validity of the pressure-release boundary assumption for the open-air surface. However, the validity of accounting for propagation absorption in the water with Eq. 3.3 may still be in question, particularly above 135 kHz. The acoustic absorption is also the primary characteristic necessary for developing analytical acoustic propagation models for the tank environment.

The spatially averaged acoustic absorption coefficient was applied to two finite-impedance boundary models that have proven to be potential candidates for modeling the tank. The derivation of these models follows the formulation for a rectangular parallelepiped acoustic water waveguide by Pierce [43] and Novak [53] and provide reasonable agreement to measured data. The Pierce model was also compared to the ORCA model in order to assess the potential for modeling a scaled open-ocean environment within the tank and shows promise. This evaluation provides an initial validation of potential capability of modeling sound propagation in the tank.

5.1 Future Work

The acoustic characterization of the tank provides an understanding of the position and frequency-dependent response, reverberation, and absorption. This method may be applied to future adaptations of the tank environment including an evaluation of absorption from artificial seabeds applied to the tank bottom. Measured absorption values provide an estimate of the acoustic transmission loss and lay a strong foundation for a precise finite-impedance boundary acoustic propagation model. Future work can expand on these initial findings through more development and evaluation of the Novak, Pierce, and ORCA models and provide refinement to these models. Development of the models could include exact implementation of source strength and an appropriate pressure amplitude term as well as the evaluation of ray-tracing models. Further development could also be made in evaluating improved acoustic propagation absorption models, that are more accurate particularly above 100 kHz within tank environments.

The measured absorption of the anechoic panels suggests the potential for additional investigations. More absorption measurements need to be taken to improve the evaluation of the anechoic nature of the side walls under variable conditions such as temperature. With further measurements more precisely defined frequency bands and physical source-receiver locations at which the tank behaves more like an open ocean with no lateral reflections may be obtained. Additional ideas for improving the absorption of the panels include building a frame to allow additional space between the anechoic panels and the acrylic walls, adding another absorption layering, or evaluating different anechoic materials, such as active ultrasonic absorbers. An absorption coefficient closer to 1 should be sought after for making a better scale-model version of open-ocean applications. The degree to which the tank approximates an open ocean may be evaluated by further comparison with ORCA, a well accepted ocean normal-mode model.

Another future topic of interest is the evaluation of the generalization error of machine learning (ML) algorithms under sound speed variability in passive SONAR application as shown in Fig. 5.1.

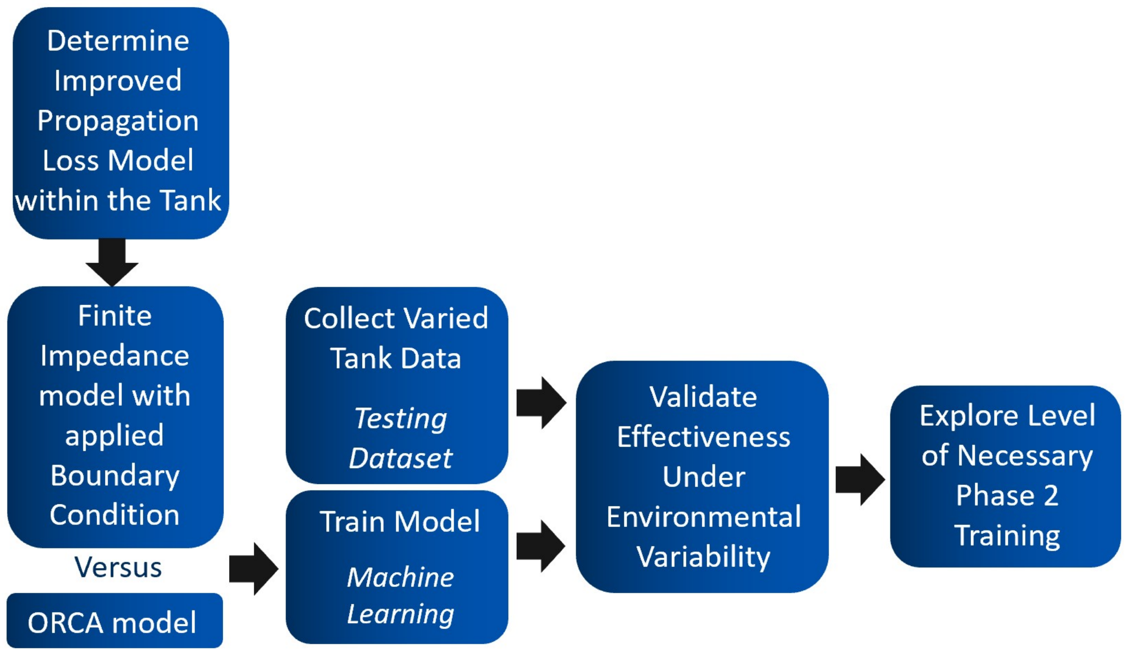

Determine Improved Propagation Loss Model within the Tank
Finite Impedance model with applied Boundary Condition
Versus
ORCA model
Collect Varied Tank Data
Testing Dataset
Train Model
Machine Learning
Validate Effectiveness Under Environmental Variability
Explore Level of Necessary Phase 2 Training

Figure 5.1 Flow chart laying out potential important future work involving applying the initial findings of this thesis in the development of models for obtaining synthetic training data. The synthetic data can be used for training machine learning models for evaluating the generalization of the machine learning model under variable to better understand necessary refinement training.

This evaluation can provide a valuable assessment of the level of retraining necessary for re-calibration of ML underwater acoustic models. This assessment is beneficial for addressing the ever changing environmental characteristics and complex nature of the open ocean and acoustic propagation.

Appendix A

List of Data

For organization and clarity and for future reference, a detailed list of data used for this thesis is provided. This list includes the data which were used to generate figures, where the data can be found, and references some of the associated code(See Appendix B) used to process said data. This appendix does not document all that is included in the logfile documentation or the LabJournal documents since a copy of each reside in the file for each measurement on the "W:Underwater" shares drive. However, each measurement has a .txt log file (seen in Appendix A.1 for an example) (following the recommendation given by Curtis [28]) that contains all information mentioned in Sec. C.4.2 along with other measurement chain components used and their settings.

All of the raw data can be found on the shares drive in the folder W:\uw-measurements-tank, where W: represents the shares/Acoustics/underwater drive, (organized by year and date) as well as a backup of the specific data sets used for this thesis in W:\Vongsawad\Data.

A.1 Log Files

The measurement parameters recorded for each individual log file contains the information seen in the example log file below. This log file example comes from scan6 measurement taken 2021-09-27 and is provided to better understand the data that is collected in each measurement.

```
Log file for ID000_000.bin (Single-Precision)


09/30/21

14:36:18

Temp: 22.77 deg C

Input Device M/N: USB-TC01 S/N: 1F2CC9E

Temp: 20.52 deg C

Input Device M/N: USB-TC01 S/N: 1F2CD84



Temperature: 20.325066 deg C and Depth: 0.567989 m

Temperature: 20.244476 deg C and Depth: 0.555501 m


Card: 4

Channel: 0


Signal Configuration:

Generated Chirp (linear) from 50000.00 Hz to 100000.00 Hz chirp

Total length: 1200000   Leading 0's: 200000

Signal length: 500000   Trailing 0's: 500000
```

```
Number of channels loaded: 1.

Loaded into channels 0.000000

.

Sampling frequency: 1000000 Hz.


Forward Step Configuration:

Number of averages: 1


Output Settings:

Reused Previously Loaded Data (T/F):

F

Card 0

Enabled: TRUE  Gain: 1000 mV

Enabled: FALSE  Gain: 1000 mV

Enabled: FALSE  Gain: 1000 mV

Enabled: FALSE  Gain: 1000 mV

Card 1

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV

Card 2

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV
```

```
Enabled: FALSE  Gain: 300 mV

Card 3

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV

Enabled: FALSE  Gain: 300 mV



Input Settings:

Card 0

Enabled: TRUE   Range: +/-5 V   Termination: Low

Enabled: TRUE   Range: +/-10 V   Termination: High

Enabled: FALSE   Range: +/-10 V   Termination: High

Enabled: FALSE   Range: +/-10 V   Termination: High


Card 1

Enabled: FALSE   Range: +/-10 V   Termination: High

Enabled: FALSE   Range: +/-10 V   Termination: High

Enabled: FALSE   Range: +/-10 V   Termination: High

Enabled: FALSE   Range: +/-10 V   Termination: High



Sampling Frequency: 1000000 (Hz)

Number of samples: 1200000
```

```
TR IR Calculation:



Focus Step Configuration:



Comments:

Source: 0.410,2.130,0.250,0.000,0.000,0.000,0.000,

0.000,0.000,0.000,0.000,0.000


Receiver: 0.410,1.830,0.250,0.000,0.000,0.000,0.000,

0.000,0.000,0.000,0.000,0.000


Water Level: 0.500000 m

Aegir Transducer: BK8103-3249189

Ran Transducer: BK8103-3249190
```

A.2 Simulation Data

The simulated data discussed and shown in Sec. 3.2.1 were created by algorithms developed to provide a test case of how the frequency deconvolution performs. The algorithms developed can be reviewed in Appendix B.2.4. These data were used in order to obtain Figures 3.11, 3.8, 3.9, 3.10, 3.4, 3.5, 3.6, 3.7, 3.12, 3.14.

A.3 Primary Measured Data

Linear swept-sine signals were generated for bandwidths 5-10 kHz, 10-50 kHz, 50-100 kHz and 100-500 kHz and were sampled at 250 kHz, 500 kHz, 1 MHz, and 1 MHz respectively with a signal length of 0.5 s led and followed by 0.5 s of zeros. The generated signals were output at 1 V for each band except for the 100-500 kHz band which output at 3 V in order to obtain better SNR (signal-to-noise-ratio). For these measurements, a Brüel & Kjær 8103 phase matched hydrophone was used as source and receiver for each bandwidth except the 100-500 kHz band which used a Teledyne Reson TC4038 hydrophone for both transmitting and receiving.

Scans used to characterize the acrylic walls of the tank are as follows for each band of interest.

- 5-10kHz: 2021-09-13/2021-09-14_scan2
- 10-50kHz: 2021-09-13/2021-09-15_scan7
- 50-100kHz: 2021-09-13/2021-09-15_scan9
- 100-500kHz: 2021-09-17B\2021-09-17_scan2

Scans used to characterize and evaluate the benefit of the anechoic panels within the tank are as follows for each band of interest.

- 5-10kHz: 2021-09-27_scan3
- 10-50kHz: 2021-09-27_scan4
- 50-100kHz: 2021-09-27_scan6
- 100-500kHz: 2021-09-22_scan4

Measurement identification numbers from the above scans used particularly for generating results and figures in this thesis (esp. in Ch. 4) come from the following ID's:

- ID028_000
- ID034_000
- ID084_000
- ID086_000
- ID109_000
- ID111_000
- ID112_000
- ID121_000
- ID130_000
- ID138_000
- ID140_000
- ID190_000
- ID196_000
- ID364_000
- ID373_000
- ID607_000
- ID616_000

Reverberation time and absorption data from the full scans of 729 source-receiver positions within the tank from 5-500 kHz were saved in .xls documents in their associated measurement folders mentioned above. This includes statistical analysis of this data. The data files are named as follows.

Reverberation time and absorption data for the acrylic walled tank:

- AbsorbAcrylic5k-10k.xlsx
- AbsorbAcrylic10k-50k.xlsx
- AbsorbAcrylic50k-100k.xlsx
- AbsorbAcrylic100k-500k.xlsx

Reverberation time and absorption data for the acrylic walled tank accounting for propagation absorption:

- AbsorbAcrylicProp5k-10k.xlsx
- AbsorbAcrylicProp10k-50k.xlsx
- AbsorbAcrylicProp50k-100k.xlsx
- AbsorbAcrylicProp100k-500k.xlsx

Reverberation time and absorption data for the anechoic walled tank

- AbsorbPanels5k-10k.xlsx
- AbsorbPanels10k-50k.xlsx
- AbsorbPanels50k-100k.xlsx
- AbsorbPanels100k-500k.xlsx

Reverberation time and absorption data for the anechoic walled tank accounting for propagation absorption

- AbsorbPanelsProp5k-10k.xlsx
- AbsorbPanelsProp10k-50k.xlsx
- AbsorbPanelsProp50k-100k.xlsx
- AbsorbPanelsProp100k-500k.xlsx

All data contained in this section represents the bulk of the data obtained for this thesis and was used for generating the figures found in Ch. 4.

A.4 Validating Measurements

The data used to provide Figure 3.3 comes from a measurement taken 2021-03-23 (W:\Vongsawad \Data\2021-03-23). The particular measurement was "scan10" used to validate the time gating and impulse response methods discussed in Sec. 3.1.3 and 3.2.1. For this measurement, the original four anechoic panels lined the y-max side of the tank and this data was specifically collected for evaluating the measurement methodology and data processing techniques.

A.5 Noise Measurements

Data collected to measure the ambient noise within the water tank environment were taken on 2021-06-28. The measurements specifically span scans 19-28 and can be found in the file path W:\Vongsawad\Data\2021-06-28. These data were used in generating Figs. 4.27, 4.28, and 4.29.

A.6 Data Confirming the Speed of Sound

Speed of sound data was collected on 2021-08-06 and is in the folder W:\Vongsawad\Data\2021-08-06B. The measurements were taken at both a central depth and width location in the tank with the purpose of evaluating over range in the tank. Scan1 was the particular measurement used for this evaluation of the speed of sound and this scan was sampled at 40 MHz in order to provide high precision when measuring such high speeds over such a small range. These data were used in generating Fig. 4.26.

Appendix B

Python Algorithms Developed

The algorithms developed for this thesis are given below. Sections are titled for the functionality of the algorithm, and a brief description of the use of said algorithm is provided. Algorithms are all available in a GIT repository shared by the research group.

B.1 Loading and Parsing Data

B.1.1 Bin File Data

The first algorithm created was ESAUdata.py, which reads in the data from binary files for a scan consisting of the generated signal, generated calibration signal, calibration recording, and four channels of recorded data from Ch0-Ch3.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 14 19:55:16 2020

This function is to load in the data from scans easily and output the desired
information needed. With any number of channels desired.

@author: cvongsaw
```

```python
"""

def binfileload(path, IDname=None, IDnum=None, CHnum=None, N=-1, Nstart=0):
    """
    Loads in a bin file generated by AFR

    Loads float32 single-precision, little-endian binary files without header
    information.

    This function returns NumPy datatypes, so make sure you are importing
    NumPy in your base scripts.

    Parameters
    ----------
    path : str
        The path to the file, do not include trailing separator. If the next three
        values are not input, it will assume that this is just a file you want to load.
    IDname : str, optional
        Root test name selected in AFR to save the files, e.g. 'ID'
    IDnum : int, optional
        Test number as recorded by AFR, e.g. 1
    CHnum : int, optional
        Channel number as recorded by AFR, e.g. 0
    N : int, optional
        Number of samples to read. Default is entire file.
    Nstart: int, optional
        Number of samples to offset from beginning of file. Default beginning.

    Returns
    -------
    x : ndarray of float
        The full array of data is returned

    Notes
    -------
    Author: David Van Komen (david.vankomen@gmail.com)

    Last Modified: 02/18/20

    Based on: binfileload.m by Kent Gee
    """
    import os
    from pathlib import Path
    import numpy as np

    if IDname == None and CHnum == None and IDnum == None:
        # we're going to use just the filename if it's ever not zero
        filename = Path(path)
        if not filename.exists():
            raise FileNotFoundError(f"The file '{filename}'' does not exist!")
    elif IDname != None and CHnum != None and IDnum != None:
        # generate the filename based on input parameters
        filename = Path(path)
        filename = filename / f"{IDname}{IDnum:03.0f}_{CHnum:03.0f}.bin"
        if not filename.exists():
            raise FileNotFoundError(
                f"The file '{str(filename)}'' governed by these inputs doesn't exist.")
    else:
        raise Exception(
            "Sorry, but if you're going to use one of the optional " +
            "filename inputs, you need to use them all.")

    # open the file as our binary file type
    with open(filename, 'rb') as binary_file:

        # fix where we start reading
        # convert Nstart to bytes instead of bits
        Nstart = Nstart * 4

        # move the binary file to where we want to start reading
        binary_file.seek(Nstart)

        # now, we get the handle for the actual data that's left
        data = binary_file.read()

        # grab the data from the buffer
        x = np.frombuffer(data, dtype=np.float32, count=N)

        binary_file.close()
    return x
# end binfile load
```

```python

def ESAUdata(path,desire=[0],channels=[0],N = 450e3,Ncal = 450e3):
    """
    Parameters
    ----------
    path:       string;
                file path name
    desire:     list;
                Desired scan positions to analyze. Should be input as a list of
                ordered scans. Defaults to only the first measurements data.
    channels:   list, Optional;
                Input channels used (channel numbers for recording data listed)
                Default to [0], being only the first channel on the spectrum cards
                [0,1] would represent the first two channels, [0,2] would represent
                the first and third channel.
    N:          float, Optional;
                Number of samples. Defaults to 450 kSamples for an fs of 150 kHz
                and trec of 3 seconds
    Ncal:       float, Optional;
                Number of samples in calgen.bin. Defaults to 450 kSamples for
                an fs of 150 kHz and trec of 3 seconds

    Returns
    -------
    gen:        float;
                generated signal for each channel outputting signal
    calgen:     float;
                generated calibration signal when selected different from gen
    cal:        float;
                calibration measurement for each channel receiving
    ch0:       float;
                all recorded scans desired from ch0
    ch1:       float;
                all recorded scans desired from ch1
    ch2:       float;
                all recorded scans desired from ch2
    ch3:       float;
                all recorded scans desired from ch3


    Notes
    -----
```

```
Author: Cameron Vongsawad

Does not currently allow for taking in second chassis daisy chained to first.

Now allows for scan without a calibration measurement by checking if a
calibration file exists before loading cal file. Else cal = zeros and errors
message about missing cal measurement is printed.
cal file format updated from cal.bin or cal (1).bin to cal_000.bin. Allows
for both potential options if referring to older measurements. It also allows
for nonsequential channel calibration measurements.

Allows for non scan measurements and when the file does not contain
a generated signal file. Also allows for agenerated calibration signal
different than the generated signal.

No longer needs byuarglib since binfileload was copied into this file and
is called locally.


last modified 04/06/2021
"""
import numpy as np
import os.path as check
import warnings
##################################################
#load generated signal and calibration measurement
##################################################
print('loading data...')

if N == None:
    N = 450e3
if Ncal == None:
    Ncal = 450e3

isFile_gen = check.isfile(path+ '/signal_out.bin')
if isFile_gen == True:
    gen = binfileload(path + '/signal_out.bin')
else:
    gen = np.empty(int(N),dtype = float)
    print('')
    print('Warning: No generated file found. gen = empty')

isFile_cal = check.isfile(path+ '/calgen.bin')
if isFile_cal == True:
    calgen = binfileload(path + '/calgen.bin')
    print('')
    print('Calibration Signal found Different from Generated Signal')
else:
    calgen = np.empty(int(Ncal),dtype = float)
    print('')
    print('Warning: No generated calibration file found. Calibration'
          +' performed same as gen or not at all. calgen = empty')


isFile0 = check.isfile(path+ '/cal_000.bin') or check.isfile(path+ '/cal.bin')
isFile1 = check.isfile(path+ '/cal_001.bin') or check.isfile(path+ '/cal (1).bin')
isFile2 = check.isfile(path+ '/cal_002.bin') or check.isfile(path+ '/cal (2).bin')
isFile3 = check.isfile(path+ '/cal_003.bin') or check.isfile(path+ '/cal (3).bin')

isFile = [isFile0,isFile1,isFile2,isFile3]

#load calibration file for each channel recorded into 1 of 4 columns in the
#array for the 4 channels allowed with the spectrum cards.
if isFile_cal == True:
    cal = np.empty((len(calgen),len(isFile)),dtype = float)
else:
    cal = np.empty((len(gen),len(isFile)),dtype = float)
#for idx,ch in enumerate(channels):
for idx in range(len(isFile)):
    if isFile[idx] == True:
        if check.isfile(path+ f'/cal_00{idx}.bin') == True:
            cal0 = binfileload(path + f'/cal_00{idx}.bin')
            cal[:,idx] = cal0
        elif check.isfile(path+ f'/cal ({idx}).bin'):
            cal0 = binfileload(path + f'/cal ({idx}).bin')
            cal[:,idx] = cal0
        elif check.isfile(path+ '/cal.bin') == True:
            cal0 = binfileload(path + '/cal.bin')
            cal[:,idx] = cal0
    else:
        #cal[:,ch] = np.zeros(len(gen),dtype = float)
        print('')
        print(f'Warning: no ch{idx} calibration file found')

if (cal ==  np.empty((len(gen),4),dtype = float)) is True:
    print('')
    print('Warning: recording error: calibration not recorded, file is empty')


"""
This is the old version of the cal file loading code. This did not allow for
measurements to be taken on unconsecutive channels or without first using ch0
if isFile0 == True:
    if check.isfile(path+ '/cal_000.bin') == True:
        cal0 = binfileload(path + '/cal_000.bin')
        cal[:,0] = cal0
    else:
        cal0 = binfileload(path + '/cal.bin')
        cal[:,0] = cal0
else:
    cal0 = np.zeros(len(gen),dtype = float)
    print('')
    print('Warning: no ch0 calibration file found')

if (cal0 == np.zeros(len(gen),dtype = float)) is True:
    print('')
    print('Warning: recording error: calibration not recorded, file is empty')

if isFile1 == True:
    if check.isfile(path+ '/cal_001.bin') == True:
        cal1 = binfileload(path + '/cal_001.bin')
        cal[:,1] = cal1
    else:
        cal1 = binfileload(path + '/cal (1).bin')
        cal[:,1] = cal1
if isFile2 == True:
    if check.isfile(path+ '/cal_002.bin') == True:
        cal2 = binfileload(path + '/cal_002.bin')
        cal[:,2] = cal2
    else:
        cal2 = binfileload(path + '/cal (2).bin')
        cal[:,2] = cal2
if isFile3 == True:
    if check.isfile(path+ '/cal_003.bin') == True:
        cal3 = binfileload(path + '/cal_003.bin')
        cal[:,3] = cal3
    else:
        cal3 = binfileload(path + '/cal (3).bin')
        cal[:,3] = cal3
        """
#load all scan binfiles
ch0 = np.empty((len(gen),len(desire)))
ch1 = np.empty((len(gen),len(desire)))
ch2 = np.empty((len(gen),len(desire)))
ch3 = np.empty((len(gen),len(desire)))
#pull only the "desire" values and their index value from the list
#to populate array
#to view channel 0
if 0 in channels:
    for idx,ich in enumerate(desire):
        ch0[:,idx] = binfileload(path,"ID",ich,0)
#to view channel 1
if 1 in channels:
    for idx,ich in enumerate(desire):
        ch1[:,idx] = binfileload(path,"ID",ich,1)
#to view channel 2
if 2 in channels:
    for idx,ich in enumerate(desire):
        ch2[:,idx] = binfileload(path,"ID",ich,2)
#to view channel 3
if 3 in channels:
    for idx,ich in enumerate(desire):
        ch3[:,idx] = binfileload(path,"ID",ich,3)

return gen, calgen, cal, ch0, ch1, ch2, ch3
```

B.1.2 Scan Position Data

The following algorithm, ESAUpose.py, is used for reading the .txt file including all positions within a grid scan using ESAU's UR10e motion control feature. These positions of both Ægir and Rán imported as a list of tuples as well as a list of range ristances between both Ægir and Rán.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 14 18:56:51 2020

This Function is for loading in ESAU scan positions to plot and use. It searches
for files called scan_positions.txt in the file path. This name convention is
what is used when using ESAU-Motion-UniversalRobots.

@author: cvongsaw
"""

def ESAUpose(path,desire = [0],plot = False,Acal = (0.6,2.14,0.3),
             Rcal = (0.6,2.06,0.3)):
    """
    Parameters
    ----------
    path:   string;
            file path name
    desire: list;
            desired scan positions. Should be input as a list of ordered scans
            this defaults to the first position if not specified otherwise in
            a list.
    plot:   Boolean {True or False}, optional;
            Default is False which does NOT returns the individual scan plots
            True returns individual scan plots with associated Source and
            Receiver positions as well as range distance in meters.
    Acal:   Tuple, Optional;
            (x,y,z) position of the AEgir calibration measurement
            Default Acal = (0.6,2.14,0.3)
    Rcal:   Tuple, Optional;
            (x,y,z) position of the Ran calibration measurement
            Default Rcal = (0.6,2.06,0.3)
    Returns
    -------
    A:      list;
            List of AEgir positions for each individual scan
    R:      list;
            List of Ran positons for each individual scan
    dd:     ndarray;
            range distance between Aegir and Ran positions for the desired
            scan positions.

    Notes
    -----
    Author: Cameron Vongsawad

    The code runs two options, the first for a scan with only one source and
    receiver position (a single measurement) and the other with more than
    one (aka an actual "scan")

    The code can also check if there is a scan_positions.txt file, if there is
    none, the code gives back zeros and a warning.

    last modified 2/3/2021
    """

    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.pylab as pylab
    params = {'legend.fontsize': 24,
               'figure.figsize': (15, 10),
              'axes.labelsize': 28,
              'axes.titlesize':29,
              'axes.titleweight':'bold',
              'xtick.labelsize':24,
              'ytick.labelsize':24,
              'lines.linewidth':3}
    pylab.rcParams.update(params)
    import os.path as check

    isFile_gen = check.isfile(path+'/scan_positions.txt')
    if isFile_gen == True:
        print('loading scan positions...')
        ################################################
        #load in scan positions as A(x,y,z) and R(x,y,z)
        ################################################
        pos = np.loadtxt(path+'/scan_positions.txt')

        ################################################
        #Load and plot positions of a SINGLE measurement
        #of single source & receiver positions
        ################################################
        if len(pos) == 24:
            a = (pos[0],pos[1],pos[2])
            r = (pos[12],pos[13],pos[14])
            A = a
            R = r
            ##########################################################
            #create x,y,and z arrays of scan positions for 3D plotting
            #and highlights positions used for a specific scan
            ##########################################################
            print('organizing scan positions...')
            xA,yA,zA = np.array([]),np.array([]),np.array([])
            xR,yR,zR = np.array([]),np.array([]),np.array([])

            xA = A[0]
            yA = A[1]
            zA = A[2]
            xR = R[0]
            yR = R[1]
            zR = R[2]
            ##################################################################
            #calculate all range distances for the full list of scan positions
            ##################################################################
            d = np.sqrt( (xA-xR)**2 + (yA-yR)**2 + (zA-zR)**2 )
            #Plot Scan Grid
            if plot == True:
                print('plotting scan position...')
                #plot scan positions
                from mpl_toolkits import mplot3d
                scan = plt.figure()
                ax = plt.axes(projection ="3d")
                ax.scatter3D(xA, yA, zA, color = "green")
                ax.scatter3D(xR, yR, zR,color = "red")
                ax.scatter3D(Acal[0],Acal[1],Acal[2], color = "orange",marker = "^")
                ax.scatter3D(Rcal[0],Rcal[1],Rcal[2], color = "orange",marker = "^")
                ax.scatter3D(A[0],A[1],A[2],color = "blue",marker = 's',
                             linewidths = 10)
                ax.scatter3D(R[0],R[1],R[2],color = "blue",marker = 's',
                             linewidths = 10)
                ax.set_xlabel('X (m)')
                ax.set_ylabel('Y (m)')
                ax.set_zlabel('Z (m)')
                ax.set_xlim(0,1.22)
                ax.set_ylim(0,3.66)
                ax.set_zlim(0,0.91)
                ax.set_title(f'S{A}, R{R}, d={round(d,3)}m')
            #########################################################
            #Only save range distances for the desired scan positions
            #########################################################
            dd = d

            return A, R, dd
        #####################################################
        #Load and plot positions of a scan of measurements
        #w/ multiple source and multiple receiver positions
        #####################################################
        else:
            A = []
            a = np.zeros(3)
            R = []
            r = np.zeros(3)
            for i in range(len(pos[:,0])):
                a = (pos[i,0],pos[i,1],pos[i,2])
                r = (pos[i,12],pos[i,13],pos[i,14])
                A.insert(i,a)
                R.insert(i,r)
            ##########################################################
            #creaty x,y,and z arrays of scan positions for 3D plotting
            #and highlights positions used for a specific scan
            ##########################################################
            print('organizing scan positions...')
            xA,yA,zA = np.array([]),np.array([]),np.array([])
            xR,yR,zR = np.array([]),np.array([]),np.array([])
            for i in range(len(A)):
                xA = np.append(xA,A[i][0])
                yA = np.append(yA,A[i][1])
                zA = np.append(zA,A[i][2])
                xR = np.append(xR,R[i][0])
                yR = np.append(yR,R[i][1])
                zR = np.append(zR,R[i][2])
            ##################################################################
            #calculate all range distances for the full list of scan positions
            ##################################################################
            d = np.empty(len(A))
            for i in range(len(A)):
                d[i] = np.sqrt( (xA[i]-xR[i])**2 + (yA[i]-yR[i])**2 +
                    (zA[i]-zR[i])**2 )
            #Plot Scan Grid
            if plot == True:
                print('plotting scan positions...')
                for i in desire:
                    #plot scan positions
                    from mpl_toolkits import mplot3d
                    scan = plt.figure()
                    ax = plt.axes(projection ="3d")
                    ax.scatter3D(xA, yA, zA, color = "green")
                    ax.scatter3D(xR, yR, zR,color = "red")
                    ax.scatter3D(Acal[0],Acal[1],Acal[2], color = "orange",
                                 marker = "^",linewidths = 4)
                    ax.scatter3D(Rcal[0],Rcal[1],Rcal[2], color = "orange",
                                 marker = "^",linewidths = 4)
                    ax.scatter3D(A[i][0],A[i][1],A[i][2],color = "blue",
                                 marker = 's',linewidths = 10)
                    ax.scatter3D(R[i][0],R[i][1],R[i][2],color = "blue",
                                 marker = 's',linewidths = 10)
                    ax.set_xlabel('X (m)')
                    ax.set_ylabel('Y (m)')
                    ax.set_zlabel('Z (m)')
                    ax.set_xlim(0,1.22)
                    ax.set_ylim(0,3.66)
                    ax.set_zlim(0,0.91)
                    ax.set_title(f'S{A[i]}, R{R[i]}, d={round(d[i],3)}m')
            ##########################################################
            #Only save range distances for the desired scan positions
            ##########################################################
            dd = np.empty(len(desire))
            for idx,i in enumerate(desire):
                dd[idx] = d[i]

            return A, R, dd
    else:
        print('')
        print('Warning: no scan position file found')
        A = [0,0,0]
        R = [0,0,0]
        dd = 0
        return A, R, dd
```

B.1.3 Measurement Parameter Data

The final major data loading related algorithm, readlogFile.py, was developed by Corey Dobbs in order to read in data from .txt logfiles from each scan. This algorithm provides individual measurement details recorded by ESAU for each measurement such as sampling rate, signal length, signal bandwidth, water temperature, water depth, etc.

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 10 15:11:18 2021

@author: Corey Dobbs
"""


def readLogFile(filename,location):
    """
    Code Description:
        This code takes the log file output by ESAU and extracts important details.
        Details described in Returns section.
        The bandwidth, source and receiver positions, sampling frequency, water
        temperature, and water depth are all printed at the end of the code.
        Relevant variables are listed as returns.



    Parameters
    ----------
    filename : String
        The name of the file (type .txt) from which you will pull the experiment
        parameters. Start and end with quotes.
        Ex: 'ID001_001log.txt'

    location : String
        Directory of file that you want to pull
        Ex: 'D:/uw-acoustics-research/uw-meas-codes/underwater-measurements/analysis/'
        IMPORTANT: Check direction of slashes

    Returns
    -------
    freqMin : Float
        The lowest frequency on the frequency range of the sweep. (Hz)
    freqMax : Float
        The highest frequency on the frequency range of the sweep. (Hz)
    tempWater : Float
        Temperature of the water at the time of measurement (degrees Celsius)
    fs : Float
        Sampling frequency of measurement (Hz)
    leadingZeros : Float
        leading zeros before the measurement starts (s)
    signalDuration : Float
        Length of signal (s)
    trailingZeros : Float
        trailing zeros after the signal has been sent (s)
    measurementDuration : Float
        Length of Measurement (s)
    hWater : Float
        Height/depth of water in tank at time of measurement, measured in meters
        from the bottom of the tank
    xSource : Float
        X-position of source (m)
    ySource : Float
        Y-position of source (m)
    zSource : Float
        Z-position of source (m)
    xRec : Float
        X-position of receiver (m)
    yRec : Float
        Y-position of receiver (m)
    zRec : Float
        Z-position of receiver (m)

    Notes
    -------
    Created by Corey Dobbs

    Last updated 4/8/2021
    """

    #Call desired directory
    #import sys
    #sys.path.insert(1,location)

    mylines = []
    with open(location+filename,"rt") as myfile:
        for myline in myfile:
            mylines.append(myline.rstrip('\n'))


    #Find receiver position
    substrR = "Receiver: "
    for line in mylines:            # string to be searched
        index = 0                     # current index: character being compared
        prev = 0                      # previous index: last character compared
        while index < len(line):      # While index has not exceeded string length,
            # set index to first occurrence of substring
            index = line.find(substrR, index)
            if index == -1:             # If nothing was found,
                break                     # exit the while loop.
            receiverPos = "(" + line[index+len(substrR):index+len(substrR)+17] + ")"

            prev = index + len(substrR)       # remember this position for next loop.
            index += len(substrR)      # increment the index by the length of substr.
                                        # (Repeat until index > line length)

    #Find source position
    substrS = "Source: "
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrS, index)
            if index == -1:
                break
            sourcePos = "(" + line[index+len(substrS):index+len(substrS)+17] + ")"

            prev = index + len(substrS)
            index += len(substrS)


    #Find water depth
    substrD = "Level: "
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrD, index)
            if index == -1:
                break
            waterDepth = line[index+len(substrD):index+len(substrD)+10]

            prev = index + len(substrD)
            index += len(substrD)


    #Find sampling frequency
    substrF = "Frequency: "
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrF, index)
            endIndex = line.find("(Hz)")
            if index == -1:
                break
            samplingFreq = line[index+len(substrF):endIndex] + "Hz"

            prev = index + len(substrF)
            index += len(substrF)


    #Find bandwidth
    substrBmin = "from "
    substrBmax = "to"
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrBmin, index)
            if index != 0 and index != -1:
                endIndex = line.find('.00')
                fmin = line[index+len(substrBmin):endIndex]
                nextIndex = line.find(substrBmax,index)
                endIndex2 = line.index('.00',nextIndex)
                fmax = line[nextIndex+3:endIndex2]
                break
            if index == -1:
                break

            fmin = line[index+len(substrBmin):endIndex]

            prev = index + len(substrBmin)
            index += len(substrBmin)
    bandwidth = fmin + "-" + fmax + " Hz"

    #Find water temperature
    substrT = "Temp: "
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrT, index)
            if index == -1:
                break
            waterTemp = line[index+len(substrT):index+len(substrT)+8]

            prev = index + len(substrT)
            index += len(substrT)

    #Find Leading 0's
    substrLead = "Leading 0's: "
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrLead, index)
            endIndex = line.find("  Signal length:")
            if index == -1:    # If nothing was found,
                break           # exit the while loop.
            leading = line[index+len(substrLead):endIndex]

            prev = index + len(substrLead)# remember this position for next loop.
            index += len(substrLead)# increment the index by the length of substr.
                                     # (Repeat until index > line length)

    #Find signal length
    substrL = "length: "
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrL, index)
            endIndex = line.find("   Trailing")
            if index == -1:            # If nothing was found,
                break                    # exit the while loop.
            signalLength = line[index+len(substrL):endIndex]

            prev = index + len(substrL)       # remember this position for next loop.
            index += len(substrL)      # increment the index by the length of substr.
                                 # (Repeat until index > line length)

    #Find Trailing 0's
    substrTrail = "Trailing 0's: "
    for line in mylines:
        index = 0
        prev = 0
        while index < len(line):
            index = line.find(substrTrail, index)
            if index == -1:     # If nothing was found,
                break            # exit the while loop.
            # no endIndex, needs to go to the end of line
            trailing = line[index+len(substrTrail):]

            prev = index + len(substrTrail)# remember this position for next loop.
            index += len(substrTrail)# increment the index by the length of substr.
                                    # (Repeat until index > line length)

    print("Bandwidth: ",bandwidth,'\n',"Water Temp: ",waterTemp,'\n',
          "Source Position: ",sourcePos,'\n',"Receiver Position: ",receiverPos,
          '\n',"Sampling Freq: ",samplingFreq,'\n',"Water Height: ",waterDepth)

    #Convert strings to variables (floats)
    freqMin = float(fmin)
    freqMax = float(fmax)
    tempWater = float(waterTemp[0:-2])
    fs = float(samplingFreq[0:-2])
    Ns = float(signalLength)
    leadingZeros = float(leading)/fs
    signalDuration = Ns/fs
    trailingZeros = float(trailing)/fs
    measurementDuration = leadingZeros + signalDuration + trailingZeros
    hWater = float(waterDepth[0:-1])
    xSource = float(sourcePos[1:6])
    ySource = float(sourcePos[7:12])
    zSource = float(sourcePos[13:-1])
    xRec = float(receiverPos[1:6])
    yRec = float(receiverPos[7:12])
    zRec = float(receiverPos[13:-1])

    return freqMin,freqMax,tempWater,fs,leadingZeros,signalDuration,trailingZeros,\
            measurementDuration,hWater,xSource,ySource,zSource,xRec,yRec,zRec
```

B.2 General Data Processing

B.2.1 Time-Gate Function

Each time-gating function, in TimeGate_UnderwaterTank.py, plays a role in predicting when sound arrives at a receiving transducer and effectively gating the signal with a half-Hanning window in order to remove side reflections. The first of three functions related to time-gating is the function that actually time-gates the input signal with input predicted values, while allowing for a buffer time δt on the window as described in Sec. 3.1.3.

```
def gatefunc(IR,fs,tgate,leading=0.0,tb4=0.1):
    """
    This function takes a signal and gates out undesired signal. At time
    "tgate" - "tb4" a hanning window will be applied that rapidly decays to
    zero.Any signal afterwards will be replaced with zeros. If leading zeros are
    input, then all those leading zeros will be replaced with actual zeros instead
    of noise. If the noise needs to be kept, then you must add the
    leading zeros to tgate before inputing it into the function and set
    "leading" equal to 0.0. If you have leading zeros in your signal you
    need to either add them to "tgate" or input them into "leading" or else
    this function will not work.

    Parameters
    ----------
    IR:     ndarray;
            Impulse Response or time domain signal.
    fs:     float;
            Sampling frequency of the input time domain signal. Measured in Hz.
    leading:float, optional;
            The leading zeros before the signal starts. Deaults to 0.0.
    tgate:  float;
            Amount of time in seconds from the beginning of the input IR signal
            in which the reflection of interest is arriving that needs to be
            timegated out of the signal.
    tb4:    float, optional;
            Defaults to 0.1 ms. This is the time before the reflection that the
            timegating should start to cut off any buildup to the reflected signal.
            This should also ideally be after the initial direct signal.

    Returns
    -------
    IRgate: ndarray;
            Time-gated array of the input signal.
    Notes
    -----
    Author: Cameron Vongsawad



    Last Modified: 4/1/2021
    """
    import numpy as np
    Nb4 = tb4/1000 *fs #convert to seconds and then samples before gating
    #where to start the time-gating or cutting off the signal to zero
    #start the time gating allowing everything from the beginning of ht
    start = int(leading*fs)
    fin = start + int(tgate*fs-Nb4)
    #convert time length to samples to determine the finish cutoff of ht
    #fin = int(tgate*fs*percent)
    #cut off the IR before the first reflection being index "fin"
    IRgate = np.zeros(len(IR))
    IRgate[start:fin] = IR[start:fin] #replace up to gate with original
    tbuff = tb4/2 #buffer value to determine where to apply the hanning window.
    damp = int(tbuff/1000*fs) #0.05ms of damping converted to samples
    #apply hanning window to portion of the array following original cutoff
    #this allows for the signal to more gradually ramp down to zeros.
    IRgate[fin:fin+damp] = IR[fin:fin+damp]*np.hanning(damp)
    #repopulate first half of that damping data keeping original array information
    IRgate[fin:int(fin+damp/2)] = IR[fin:int(fin+damp/2)]

    return IRgate
```

The following function estimates sound speed in water according to the formulation of either Garrett [3], Medwin [54], or Wilson [55]. Each of these simple formulations provide similar results and were developed from or have inspired many other methods for estimating the speed of sound in water [56–61]. This thesis primarily focuses on Garrett's formulation for simplicity and confirmed this estimated value through measuring the time delay via cross-correlation on range-dependent measurements. The average value of the sound speed in the tank was estimated at 1478 m/s and measured to be 1486.5 m/s. Many other methods exist for determining the sound speed or sound speed profile in water, this method was accepted in this thesis considering the small nature of the tank. Though with improved precision comes decreased error present in other calculations.

```

def uwsoundspeed(D=0.2,T=16.0,S=0.03,model='Garrett'):
    """
    Compute the Sound Speed of the Water based on the Depth, Temperature, and
    Salinity of the water according to three well known models. Garrett, Medwin
    & Kuperman or Wilson.

    Parameters
    ----------
    #Water Characteristics in the Tank#
    D:  float, optional;
        water depth (m) where 0<= D <=1000m
    T:  float, optional;
        temperature in Celcius where -2<= T <=24.5
    S:  float, optional;
        salinity where 0.030<= S <=0.042 grams salt per kg H20 (aka parts per
        thousand = ppt)
    Returns
    -------
    c:  float;
        speed of sound in water for the specified depth, temperature and salinity

    Notes
    -----
    Author: Cameron Vongsawad

    Last Modified: 4/1/2021
    """
    ##############################################################################
    ######## sound speed (m/s), Cite Garrett valid w/in +-0.2m/s ##################
    ###### appears to be accurate w/in 0.000969% of wiki value @20C ###############
    ########## effects of depth is negligible in the tank limits ##################
    ########### EQ 11.26 pg 619 Garrett "Understanding Acoustics" #################
    ##############################################################################
    if model == 'Garrett' or 'garrett' or None:
        c = 1493 + 3*(T-10) - 0.006*(T-10)**2 - 0.04*(T-18)**2 + 1.2*(S-35)- (0.01
                    *(T-18)*(S-35) + D/61)

    #medwin & Kuperman Encyclo. of Ocean Sciences 2nd ed. 2001
    if model == 'Kuperman' or 'Medwin' or 'MedwinKuperman':
        c = 1449.2 + 4.6*T - 0.055*T**2 + 0.00029*T**3 + ((1.34 - 0.010*T)*(S
                                                                        - 35)+0.016*D)

    #Christ, WenliSr., The ROV Manual 2nd ed. 2014 from simplified Wilson's 1960
    #S is in PSU which is basically equivalent to ppt
    if model == 'Wilson':
        c = 1449 + 4.6*T - 0.055*T**2 + 0.0003*T**3 + 1.39*(S - 35) + 0.017*D
    return c
```

The final time-gating algorithm from TimeGate_UnderwaterTank.py uses the method of images and ray theory in order to predict the arrival time of the first reflection off each boundary to the receiving hydrophone compared to the time of estimated direct sound arrival. The precision of this arrival time estimation depends on the estimated speed of sound and the particular frequency content of interest behaving or propagating as rays (best assumed in frequencies above the Schroeder frequency as discussed in Sec. 3.1.2).

```
def gateValue(AEgir_pose, Ran_pose, D, c=1478, Coordinate='tank',Print='True'):
    import numpy as np
    """
    Compute the first bounce reverberations of the BYU Hydroacoustics lab tank
    in order to timegate signals. Assumes a rectangular volume.

    Parameters
    ----------
    AEgir_Pose: tuple;
                AEgir TCP position (x,y,z)
    Ran_pose:   tuple;
                Ran TCP position (x,y,z,v) if 'robot' frame or (x,y,z) if 'tank'
                frame where v is the vention position (7th axis extender)

    #Water Characteristics in the Tank#
    D:  float, optional;
        water depth (m) where 0<= D <=1000m

    Coordinate: string, optional;
                Choose if cordinate system is robot frame or tank frame
                Standard is tank frame "tank"
                or robot frame inputing each robot + vention positioning "robot"
    Print: string, optional;
                Choose if you want the function to print a bunch of numbers


    Returns
    -------
    tshort: float;
            shortest time for a single reflection in seconds.
    tside:  float;
            shortest time for single reflection of side wall reflections only
            but still allowing potential for seabed and surface reflections.
    tdirect: float;
            time for direct signal to arrive based on input speed of sound.
    directpath: float;
                distance of direct path from hydrophone to hydrophone

    prints values of:
        AEgir and Ran positions
        direct sound "tdirect"
        Single bounce times:
            bottom "tb"
            H20-02 "tt"
            Side 1 "ts1"
            Side 2 "ts2"
            Front wall "tfront"
            Back wall "tback"

    Notes
    -----
    Author: Cameron Vongsawad

    This code only allows for a single tuple of length len(A)=3 and len(R)=3 or 4
    Times printed in "ms" (milliseconds), however returned values in seconds

    Last Modified: 6/1/2021

    """

        ############################################################################
        ################## function to determine time of flight for ##############
        ############# ray paths knowing tank fram positions ######################
        ############################################################################
    def pathtime(XA,YA,ZA,XR,YR,ZR):
        """
        XA, YA, ZA  : float, cartesian coordinates of AEgir
        XR, YR, ZR  : float, cartesian coordinates of Ran

        """
        ############################################################################
        ######################## main code for direct path time ##################
        ############################################################################
        directpath = np.sqrt((XA-XR)**2+(YA-YR)**2+(ZA-ZR)**2)   #direct distance eq
        tdirect = (directpath)/c

        ############################################################################
        ######################## main code for bottom bounce #####################
        ####################### determined through geometries ####################
        ############################################################################
        range_b = np.sqrt(directpath**2 - np.abs(ZA-ZR)**2)      #r bottom z-y plane
        rzA = ZA/np.sin(np.arctan((ZA+ZR)/range_b))
        rzR = ZR/np.sin(np.arctan((ZA+ZR)/range_b))
        tb = (rzA + rzR)/c                                          #bottom bounce time

        ############################################################################
        ######################### main code for top bounce #########################
        ####################### determined through geometries ######################
        ############################################################################
        ZAt = D - ZA                        #translate to looking from water surface
        ZRt = D - ZR
        range_t = np.sqrt(directpath**2 - np.abs(ZAt-ZRt)**2)      #r top  z-y plane
        rzAt = ZAt/np.sin(np.arctan((ZAt+ZRt)/range_t))
        rzRt = ZRt/np.sin(np.arctan((ZAt+ZRt)/range_t))
        tt = (rzAt + rzRt)/c                                           #top bounce time

        ############################################################################
        ######################### main code for side1 x=0 bounce ###################
        ####################### determined through geometries ######################
        ############################################################################
        range_s1 = np.sqrt(directpath**2 - np.abs(XA-XR)**2)        #r x=0 x-y plane
        rxAs1 = XA/np.sin(np.arctan((XA+XR)/range_s1))
        rxRs1 = XR/np.sin(np.arctan((XA+XR)/range_s1))
        ts1 = (rxAs1 + rxRs1)/c                                        #x=0 bounce time

        ############################################################################
        ######################### main code for side2 X=X bounce ###################
        ####################### determined through geometries ######################
        ############################################################################
        Xmax = 1.22
        XAs2 = Xmax - XA                    #translate to looking from water surface
        XRs2 = Xmax - XR
        range_s2 = np.sqrt(directpath**2 - np.abs(XAs2-XRs2)**2)    #r x=x x-y plane
        rxAs2 = XAs2/np.sin(np.arctan((XAs2+XRs2)/range_s2))
        rxRs2 = XRs2/np.sin(np.arctan((XAs2+XRs2)/range_s2))
        ts2 = (rxAs2 + rxRs2)/c                                        #x=x bounce time

        ############################################################################
        ################## main code for "front" (North) wall bounce 1 y=0 #########
        ################## using the method of images                      #########
        ############################################################################
        range_front = np.sqrt((XA-XR)**2+(YA-(-YR))**2+(ZA-ZR)**2)  #direct image
        tfront = (range_front)/c                                       #front wall time

        ############################################################################
        ################# main code for "back" (South) wall bounce 2 y=y##########
        ################# using the method of images                     #########
        ##########################################################################
        Ymax = 3.66
        range_back = np.sqrt((XA-XR)**2+(YA-(YR+Ymax))**2+(ZA-ZR)**2)  #direct image
        tback = (range_back)/c                                      #front wall time



        t = (tb,tt,ts1,ts2,tfront,tback)
        tshort = min(t)
        tside = (ts1,ts2,tfront,tback)
        tside = min(tside)
        if Print:
            print('')
            print('AEgir(source) & Ran(Receiver) tank frame coordinates:')
            print('(XA,YA,ZA)=',(XA,YA,ZA))
            print('(XR,YR,ZR)=',(XR,YR,ZR))
            print('')
            print('Single Bounce reverberation times to receiver:')
            print('direct sound t=', tdirect*10**3,'ms')
            print('bottom bounce t=', tb*10**3,'ms')
            print('H20-02 bounce t=', tt*10**3,'ms')
            print('Side 1 bounce t=', ts1*10**3,'ms')
            print('Side 2 bounce t=', ts2*10**3,'ms')
            print('Front Wall bounce t=', tfront*10**3,'ms')
            print('Back Wall bounce t=', tback*10**3,'ms')
            print('tshort=',tshort,'s')
            print('')
        return tshort,tside,tdirect,directpath


        #### for tank frame coordinates, no need to translate coordinates #########
    if Coordinate == 'tank':
        ##########################################################################
        ## Hydrophone Locations (Insert AEgir & Ran Tank coordinates (X,Y,Z) in m) #
        ##########################################################################
                        #"AEgir" Tank Frame position (X,Y,Z) TCP TC4038
        XA   = AEgir_pose[0]
        YA   = AEgir_pose[1]
        ZA   = AEgir_pose[2]
                        #"Ran" Tank Frame position (X,Y,Z) TCP TC4034
        XR   = Ran_pose[0]
        YR   = Ran_pose[1]
        ZR   = Ran_pose[2]

        tshort,tside,tdirect,directpath = pathtime(XA,YA,ZA,XR,YR,ZR)


        #### for robot frame coordinates, must translate to tank frame first #######
    elif Coordinate == "robot":
        ############################################################################
        ########## TCP Locations (insert current TCP locations in mm)##############
        #### This is in correlation with default settings for the end connector ####
        ############################################################################
                        #"AEgir" position TCP TC4038
        TCPxA   = AEgir_pose[0]
        TCPyA   = AEgir_pose[1]
        TCPzA   = AEgir_pose[2]
                        #"Ran" position TCP TC4034
        TCPxR   = Ran_pose[0]
        TCPyR   = Ran_pose[1]
        TCPzR   = Ran_pose[2]
        #Vention 7th axis positioning adjustment for y direction
        TCPvR   = Ran_pose[3]

        ############################################################################
        ######## Tank Frame Locations (insert current tank locations in mm)#########
        ######## these are directly measured values. must comment out future #######
        ######## translation of positioning if used. OR translation trumps this ####
        ############################################################################

        #convert mm positioning to m
        TCPxA = TCPxA/1000
        TCPyA = TCPyA/1000
        TCPzA = TCPzA/1000
        TCPxR = TCPxR/1000
        TCPyR = TCPyR/1000
        TCPzR = TCPzR/1000
        TCPvR = TCPvR/1000

        ############################################################################
        #translating TCP position to tank coordinate positions (/1000 for mm => m) #
        #Home position used for conversion w/end connector settings of both ########
        #"AEgir" and "Ran" measured in mm initially and then later converted to m ##
        #Home position in the tank frame for "Ran" should be measured at VR = 0 ####
        ############################################################################
        XA_TCP_home = 392.69/1000
        YA_TCP_home = 288.83/1000
        ZA_TCP_home = -59.54/1000
        XA_tank_home = 100/1000
        YA_tank_home = 2984/1000
        ZA_tank_home = 901/1000

        XR_TCP_home = 1291.6/1000
        YR_TCP_home = 132.98/1000
        ZR_TCP_home = -190.91/1000
        VR_TCP_home = 1404.7
        XR_tank_home = 1010/1000
        YR_tank_home = 541/1000
        ZR_tank_home = 721/1000
        VR_tank_home = YR_tank_home

        XA =  (XA_tank_home + (-XA_TCP_home + TCPxA) )
        YA =  (YA_tank_home + (-YA_TCP_home + TCPyA) )
        ZA =  (ZA_tank_home + (-ZA_TCP_home + TCPzA) )
        XR =  (XR_tank_home + (-XR_TCP_home + TCPxR) )
        #adjusted for Vention pos
        YR =  (YR_tank_home + (-YR_TCP_home + TCPyR) -TCPvR )
        ZR =  (ZR_tank_home + (-ZR_TCP_home + TCPzR) )

        tshort,tside = pathtime(XA,YA,ZA,XR,YR,ZR)

    return tshort,tside,tdirect,directpath
```

B.2.2 Fractional Octave Filtering

The following code (found in TankCharacterization.py) was developed primarily by Corey Dobbs following IEC 61260-1:2014 [51] in order to apply a fractional octave filter to data-sets (especially for swept-sine signals). Typically octave band and 1/3 octave band filters are used. Since this thesis deals with such high frequency content, this function allows for any fractional octave. For this thesis, a 1/25th or 1/30th octave filter was used in order to gather frequency-dependent reverberation and absorption data from swept-sine signal scans within the tank environment.

```python
def OctaveFilter(data,f0,f1,fs,frac = 1,order = 5,exact = True):
    """

    Parameters
    ----------
    data:       Ndarray;
                Sampled data that covers some bandwidth.
    f0:         float;
                Low-end frequency (Hz) of the desired bandwidth
    f1:         float;
                High-end frequency (Hz) of the desired bandwidth
    fs:         float;
                Sampling frequency of the data
    frac:       float, Optional;
                Bandwidth fraction. Examples: 1/3-octave frac=3, 1-octave frac=1
                (Default), 2/3-octave frac=3/2.
    order:      Int, Optional;
                Order of the filter. Defaults to 5.
    exact:      boolean;
                Gives option to use IEC standard for octave ratio (10**(3/10))
                or generally accepted standard of 2. Default is True. Set exact
                to False if factor of 2 is desired.

    Returns
    -------
    filt_data:  Ndarray;
                2-d array of the bandpass filtered data. Row dimensions = same
                dimensions as mid_bands. Each row is the data for a given band.
                The column dimensions are the filtered data. Ex) filt_data[0,:]
                would be all of the data for the first mid-band frequency.

    mid_bands:  Ndarray of float;
                Array of octave or fractional octave frequencies
                Note: center frequencies are based on IEC standard 61260-1
                found in equation 1 in section 5.2.1. This code defaults to the
                octave ratio 10**(3/10) as opposed to the standard ratio of 2.


    Notes
    -----
    Author: Corey Dobbs

    Apply a bandpass filter to data in order to obtain an average over an
    octave or fractional octave band centered at the middle frequencies output
    in mid_bands.

    References:
    https://scipy-cookbook.readthedocs.io/items/ButterworthBandpass.html

    https://github.com/jmrplens/PyOctaveBand/blob/
    43e65e6cfc50d0b079383fee7ba0693cd645c350/PyOctaveBand.py#L14

    TDOTOspec.m by Dr. Kent Gee at BYU, found in BYU Acoustics
    under General Signal Processing/src/Analyzing Spectra
    https://git.physics.byu.edu/acoustics

    Dr. Gee's code included this note:
    BUTTER is based on a bilinear transformation, as suggested in
    ANSI standard.  From oct3dsgn function by Christophe Couvreur, Faculte
    Polytechnique de Mons (Belgium)


    last modified 9/1/2021
    """
    import numpy as np
    import math
    import scipy.signal as sig


    #Generate Frequency Array
    if exact == True:
        G = 10**(3/10) #octave frequency ratio
        #based on IEC standard 61260-1 found in equation 1 in section 5.2.1.
    elif exact == False:
        G = 2
        #generally accepted octave frequency ratio
    fr = 1000     #reference frequency


    # Get the initial mid-band frequency
    #According to IEC standard 61260-1 section 5.4
    if frac % 2 == 0: #Even frac
        x_init = math.ceil(frac*np.log(f0/fr)/np.log(G) - 1/2)
        x_final = math.floor(frac*np.log(f1/fr)/np.log(G) - 1/2)
    else: #Odd frac
        x_init = math.ceil(frac*np.log(f0/fr)/np.log(G))
        x_final = math.floor(frac*np.log(f1/fr)/np.log(G))

    x = np.arange(x_init,x_final + 1)


    #Get mid-band frequencies and limits
    if frac % 2 != 0: #Odd frac
        mid_bands = fr*G**(x/frac)
    else: #Even frac
        mid_bands = fr*G**((2*x+1)/(2*frac))

    #Get frequency band limits
    #References codes by Kent Gee and Christophe Couvreur
    upper_limits = mid_bands*G**(1/(2*frac)) #low ends of filter
    lower_limits = mid_bands/G**(1/(2*frac)) #high ends of filter
    Qr = mid_bands/(upper_limits - lower_limits)
    Qd = np.pi/2/frac/np.sin(np.pi/2/frac)*Qr
    alpha = (1 + np.sqrt(1+4*Qd**2))/2/Qd



    #Zero mean
    data = data - np.mean(data)

    #Window, and rescaling
    w = np.hanning(len(data))
    data = data*w/np.sqrt(np.mean(w**2))


    #Use a butterworth filter on the data according to the fractional octave bands
    for i in range(len(mid_bands)):

        #Use a decimation factor to keep the sampling frequency within
        #reasonable limits.

        if mid_bands[i] < fs/20: #factor of 20 suggested as threshold for decimation
```

```
            deci_rat = np.ceil(fs/mid_bands[i]/20)  #Decimation factor
            #decdata = sig.decimate(sig.decimate(data,10),2)
        else:
            deci_rat = 1

        fsdec = fs/deci_rat #Decimated sampling rate


        W1 = mid_bands[i]/(fsdec/2)/alpha[i]
        W2 = mid_bands[i]/(fsdec/2)*alpha[i]


        b,a = sig.butter(order, [W1, W2], btype='band')

        #Rescale decimated data
        if deci_rat > 1:
            decdata = sig.resample(data, int(len(data)/deci_rat))
        else:
            decdata = data

        placeholder = sig.lfilter(b,a,decdata)

        #Interpolate back up to original length of data
        #This ensures that the output filt_data is a nxm array, where
        #n is the number of center frequencies and m is the original length of
        #the data
        if len(decdata) != len(data):
            dummy_time_act = np.arange(len(data))/fs
            dummy_time = np.arange(len(decdata))/fsdec
            placeholder = np.interp(dummy_time_act, dummy_time, placeholder)


        #This initializes the filt_data array
        if i == 0:
            filt_data = np.zeros((len(mid_bands),len(placeholder)))

        #Fill in filt_data with the filtered data held in placeholder
        for j in range(len(placeholder)):
            filt_data[i,j] = placeholder[j]

    return filt_data, mid_bands
```

B.2.3 Impulse Response with Frequency Deconvolution

The following code shows three algorithms developed within ESAUResponse.py. Each function (IR, SysResponse, and TankResponse) are used in conjunction to calculate an impulse response (IR) from a measurement through frequency deconvolution, using said impulse response (see Sec. 3.2.1) to determine an *in situ* calibration (see Sec. 3.2.2) response, and applying this calibration to measured data to determine an overall tank response.

IR is the general function to process the impulse response from a recorded signal relative to a reference or generated signal. The IR function was designed to effectively process the impulse response of a system excited by a swept-sine signal. It utilizes Wiener deconvolution to avoid division by zero through the use of a regularization parameter. This function adjusts for noise by performing the deconvolution via division in the frequency domain of the cross-spectrum by the auto-spectrum of the input signals. Options for processing in alternate but similar methods are provided. This function simply returns the impulse response in the time domain.

```python
def IR(rec,gen,fs,wiener=False,domain='f'):
    """
    Parameters
    ----------
    rec:        ndarray of float of size 1;
                time domain of the received signal. Should be real valued.
    gen:        ndarray of float;
                time domain of the generated signal. Should be real valued.
    fs:         float;
                Sampling frequency in Hz
    wiener:     Boolean {True or False}; optional;
                False (default) for using direct deconvolution instead of Wiener
                deconvolution in frequency domain. If (True), the Wiener
                deconvolution is performed. Wiener deconvolution acts as a
                regularization which helps prevent dividing by zero allowing for
                a more robust deconvolution while maintaining an account for any
                system response.
    domain:     string, Optional;
                Choice of domain performs the inverse filter in the initial step
```

```
                    in either the temporal domain ('t' or 'time' or 'temporal') or
                    in the frequency domain (default) ('f' or 'freq' or 'frequency')
                    which is equivalent to determining the the cross-spectral density
                    and the auto-spectral density for the use in the deconvolution.
                    The end deconvolution always occurs by division in frequency domain.

    Returns
    -------
    ht:             ndarray of float;
                    Real valued impulse response (IR) of a measurement.

    Notes
    -----
    Author: Cameron Vongsawad

    The IR h(t) is determined following Gemba(2014) eq. 3.3.6 scaling similar
    to a matched filter and deconvolving in order to obtain the pure delay of h(t).
    Cite: "Characterization of underwater acoustic sources recorded in reverberant
    environments with application to scuba signatures" Gemba (2014) Dissertation.
    Also see eq. 3.3.3 and 3.3.5

    This also follows the directions from Farina 2000 and Farina 2007 on IR
    from swept-sines.

    Also see eq. 1.7.1, 1.7.2, and 1.7.3 from Leishman 560 notes 2019.

    Deconvolution all in the frequency domain should be much faster computationally.

    Dr. Brian Anderson published a paper discussing Wiener deconvolution as a
    regularization parameter for deconvolution. He particularly discusses
    optimizing lambda."Time reversal focusing of high amplitude sound in a
    reverberation chamber" (2018) Willardson, Anderson, Young, Denison, Patchett.
    https://doi.org/10.1121/1.5023351

    last modified 6/30/2021
    """
    import numpy as np
    if domain == 'time' or 't' or 'temporal':
        #The time domain is a slower computation
        #rec(t)*gen(-t)) = h(t)*gen(t)*gen(-t) eq 3.3.5 solve for h(t)
        gen_flip = np.flip(gen)
        #np.convolve(gen,gen_flip) == sci.correlate(gen,gen) by def.
        #the inverse filter of the function is np.convolve(gen,gen_flip)
        #for noise at output (receiver)
        saa = np.convolve(gen,gen_flip,mode='same')
        sab = np.convolve(rec,gen_flip,mode='same')
        #The following does the same thing but via correlate:
        #import scipy.signal as sci
        #saa = sci.correlate(gen,gen,mode='same',method='auto')
        #sab = sci.correlate(rec,gen,mode='same',method='auto')

        #for noise at input (source) which is more rare
        #rec_flip = np.flip(rec)
        ##sbb = np.convolve(rec,rec_flip,mode='same')
        ##sba = np.convolve(rec_flip,gen,mode='same')


        import matplotlib.pyplot as plt
        plot = False
        #proof that this method applies what some literature refers to as
        #an inverse filter.
        if plot == True:
            plt.figure()
            plt.plot(np.abs(saa))
            plt.title('Delta Function as result of the Inverse Filter Convolution')
            plt.xlabel('time (Samples)')
            plt.ylabel('Amplitude')
            plt.grid()

        #Division in the frequency domain is a deconvolution in the time domain.
        #which gives H(f) and then ifft(H(f))=h(t)
        Sab = np.fft.fft(sab) #double-sided frequency response
        Saa = np.fft.fft(saa) #double-sided frequency response
        ##Sbb = np.fft.fft(sbb)
        ##Sba = np.fft.fft(sba)
        #f = np.fft.fftfreq(len(xcorr),d=1/fs)

    if domain == 'frequency' or 'freq' or 'f':
        #COMPUTE ALL of the deconvolution in FREQ DOMAIN instead of time domain,
        #should be faster for noise at output (receiver)
        Sab = np.conj(np.fft.fft(gen))*np.fft.fft(rec)
        Saa = np.conj(np.fft.fft(gen))*np.fft.fft(gen)
        #for noise at input (source) which is more rare
        ##Sbb = np.conj(np.fft.fft(rec))*np.fft.fft(rec)
        ##Sba = np.conj(np.fft.fft(rec))*np.fft.fft(gen)

        #f = np.fft.fftfreq(len(gen),d=1/fs)

    if wiener == True:
        print('Performing deconvolution via Wiener deconvolution'
              +'preventing dividing by zero')
        #Wiener Deconvolution deals with the near zero values which cause
        #processing noise and high frequency aliasing.
        lamb = 0.005 #scaling parameter arbitrarily chosen
        #for noise at output (receive)
        sigma = lamb*np.mean(np.abs(Saa)) #expectation or noise or SNR
        WDeconv = np.conj(Saa)*Sab/(np.abs(Saa)**2+sigma**2)
        #for noise at input (source)
        ##sigma = lamb*np.mean(np.abs(Sba)) #expectation or noise or SNR
        ##WDeconv = np.conj(Sba)*Sbb/(np.abs(Sba)**2+sigma**2)
        Deconv = WDeconv
    else:
        print('Performing deconvolution via direct division in frequency domain')
        #Perform standard deconvolution by direct division in frequency domain.
        #for noise at output (receive)
        Deconv = Sab/Saa
        #for noise at input (source)
        ##Deconv = Sbb/Sba

    #bring back to time domain with inverse fast fourier transform (IFFT)
    ht = np.real(np.fft.ifft(Deconv)) #ensure real valued as it should be
    return ht
```

The SysResponse function is used as a calibration function as discussed in Sec. 3.2. This calibration function performs deconvolution in the frequency domain to obtain a response of the measurement chain assuming small propagation losses with source and receiver positioned close. Time-gating is performed by the Time-Gate function shown in discussed in Sec. 3.1.3 and shown in Appendix B.2.1.

```


def SysResponse(cal,gen,fs,tgate=0,wiener=False,domain='f'):
    """
    Parameters
    ----------
    cal:         ndarray of float;
                 Received calibration signal. Should be real valued.
    gen:         ndarray of float;
                 Pure generated signal. Should be real valued.
    fs:          float;
                 Sampling frequency (Hz).
    tgate:       float,Optional;
                 Time of the first wall reflection determined through timegateTank.
                 This is the time we will use to determine the time of the first
                 reflection and timegate the impulse response of the calibrated
                 signal by. This input is optional if you want to timegate. If
                 not wanting to timegate the IR, leave tgate = 0 which is the default.
                 If tgate is nonzero, the IR will be gated according the input time.
    wiener:      Boolean {True or False}; optional;
                 False (default) for using direct deconvolution instead of Wiener
                 deconvolution in frequency domain. If (True), the Wiener
                 deconvolution is performed. Wiener deconvolution acts as a
                 regularization which helps prevent dividing by zero allowing for
                 a more robust deconvolution while maintaining an account for any
                 system response.
    domain:      string, Optional;
                 Choice of domain performs the inverse filter in the initial step
                 in either the temporal domain ('t' or 'time' or 'temporal') or
                 in the frequency domain (default) ('f' or 'freq' or 'frequency')
                 which is equivalent to determining the the cross-spectral density
                 and the auto-spectral density for the use in the deconvolution.
                 The end deconvolution always occurs by division in frequency domain.

    Returns
    -------
    ht:          ndarray of float;
                 Real valued impulse response (IR) of the measurement chain neglecting
                 effects of the water and tank environment through timegating only
                 direct signal with a small propagation assumption.
    t:           ndarray of float;
```

```
                    Time array for the IR h(t) in seconds (s)
    Hf:             ndarray of complex;
                    Complex two-sided Frequency Response to account for all transducer,
                    amplifier, etc. in the measurement chain.
    f:              ndarray of float
                    frequency array in (Hz)

    Notes
    -----
    Author: Cameron Vongsawad

    Measurements should be taken with source and receiver close together and
    in the center of the tank so it is easy to time gate the signal. Not too
    close that nonlinear effects occur. The callibration position should already
    be hard coded into ESAU but can be changed manually.

    The IR h(t) is determined first by timegating the signal for only direct sound
    then following Gemba(2014) eq. 3.3.6 and Farina 2000,2007 with the use of an
    inverse filter and scaling similar to a matched filter and deconvolving in
    order to obtain the pure delay of h(t).
    Cite:
    "Characterization of underwater acoustic sources recorded in reverberant
    environments with application to scuba signatures" Gemba (2014) Dissertation

    Farina (2000)
    Farina (2007)

    Often a single-sided response is desired. We find the s-sResponse
    as follows below:
        Hss = 2*Hf[0:(int(len(Hf)/2))]      #convert to single-sided FRF
        fss = f[0:int(len(f)/2)]            #convert to single-sided

    last modified 5/17/2021
    """
    import numpy as np
    from ESAUResponse import IR

    ht = IR(cal,gen,fs,wiener=wiener,domain=domain)     #IR through deconvolution
    t = np.linspace(0,len(ht)/fs,len(ht))            #time array for ht

    if tgate !=0:
        print('Timegating the IR of the signal...')
        import TimeGate_UnderwaterTank as tg
        ht = tg.gatefunc(ht,fs,tgate,tb4=0.1) #cut off wall reflections

    #calculate the FRF from the IR and obtain the associated freq array
    print('calculating the 2-sided Frequency Response...')
    #Report the double-sided time-gated FRF of the input IR
    Hf = np.fft.fft(ht)
    #Report the double-sided associated freq array
    f = np.fft.fftfreq(len(ht),d=1/fs)

    return ht,t,Hf,f
```

The TankResponse function is used to apply the calibration obtained from SysResponse to a recorded signal and provide a true response of an environment being measured accounting for any effects of the A-D and D-A system.

```


def TankResponse(rec,gen,fs,sysIR,wiener=True,domain='f'):
    """
    Parameters
    ----------
    rec:        ndarray of float;
                Received signal. Should be real valued.
    gen:        ndarray of float;
                Pure generated signal. Should be real valued.
    fs:         float;
                Sampling frequency (Hz)
    sysIR:      ndarray;
                This is the system impulse response h(t) of the whole measurment
                chain found between two close points using SystemResponse func.
    wiener:     Boolean {True or False}; optional;
                False (default) for using direct deconvolution instead of Wiener
                deconvolution in frequency domain. If (True), the Wiener
                deconvolution is performed. Wiener deconvolution acts as a
                regularization which helps prevent dividing by zero allowing for
                a more robust deconvolution while maintaining an account for any
                tank response effects.
    domain:     string, Optional;
```

```
                    Choice of domain performs the inverse filter in the initial step
                    in either the temporal domain ('t' or 'time' or 'temporal') or
                    in the frequency domain (default) ('f' or 'freq' or 'frequency')
                    which is equivalent to determining the the cross-spectral density
                    and the auto-spectral density for the use in the deconvolution.
                    The end deconvolution always occurs by division in frequency domain.

    Returns
    -------
    H_tank:         ndarray of float;
                    Complex two-sided Greens function of Frequency Response of the Tank
                    envrionment
    f:              ndarray of float;
                    Two-sided frequency array matching the frequency response H_tank

    Notes
    -----
    Author: Cameron Vongsawad

    This greens function is relative to the individual positions of the Source
    and Receiver in the tank. To use this, you will need to also run the
    SystemResponse functin to obtain the frequency response of the
    measurement chain (transducers,etc.)

    last modified 5/17/2021
    """
    import numpy as np
    from ESAUResponse import IR
    #obtain the IR of the recorded signal relative to the generated signal
    ht = IR(rec,gen,fs,wiener=wiener,domain=domain)
    #if ht.size != sysIR.size, zeropadding is necessary at the end of the smaller
    #so they are both of the same length, thus interpolating the FRF which allows
    #the components of the deconvolution to be the same size.
    if len(ht)<len(sysIR):
        #number of zeros needed for padding to obtain same size for ht
        nzeros =int(np.abs(len(ht)-len(sysIR)))
        h = np.zeros(len(ht))
        fin = int(.999*len(ht))
        damp = int(0.0005*len(ht))
        h[0:fin] = ht[0:fin] #replace up to gate with original
        #apply half-hanning window to last portion of the array this allows for
        #the signal to more gradually ramp down to zeros to be padded.
        h[fin:fin+damp] = ht[fin:fin+damp]*np.hanning(damp)
        #repopulate first half of that damping data keeping original array information
        h[fin:int(fin+damp/2)] = ht[fin:int(fin+damp/2)]
        #pad the end of the array with zeros making up for the difference
        ht0 = np.pad(h,(0,nzeros),'constant',constant_values=(0,0))
        sys = sysIR

    if len(sysIR)<len(ht):
        nzeros =int(np.abs(len(ht)-len(sysIR)))
        s = np.zeros(len(sysIR))
        fin = int(.999*len(sysIR))
        damp = int(0.0005*len(sysIR))
        s[0:fin] = sysIR[0:fin] #replace up to gate with original
        #apply hanning window to last portion of the array this allows for
        #the signal to more gradually ramp down to zeros to be padded.
        s[fin:fin+damp] = sysIR[fin:fin+damp]*np.hanning(damp)
        #repopulate first half of that damping data keeping original array information
        s[fin:int(fin+damp/2)] = sysIR[fin:int(fin+damp/2)]
        sys = np.pad(s,(0,nzeros),'constant',constant_values=(0,0))
        ht0 = ht

    if len(sysIR) == len(ht):
        sys = sysIR
        ht0 = ht

    #Obtain frequency response of both the sysIR and ht for deconvolution in freq.
    Sys = np.fft.fft(sys)
    Hf = np.fft.fft(ht0)
    f = np.fft.fftfreq(len(ht0),d=1/fs) #associated frequency array

    if wiener == True:
        print('performing deconvolution via Wiener deconvolution preventing'
              +' dividing by zero')
        #Wiener Deconvolution deals with the near zero values which cause
        #processing noise and high frequency aliasing.
        lamb = 0.005 #scaling parameter arbitrarily chosen
        sigma = lamb*np.mean(np.abs(Sys))
        WDeconv = np.conj(Sys)*Hf/(np.abs(Sys)**2+sigma**2)
        Deconv = WDeconv
    else:
        print('Performing deconvolution via direct division in frequency domain')
        Deconv = Hf/Sys #standard deconvolution is division in freq domain

    Htank = Deconv

    return Htank,f
```

B.2.4 Evaluation of the Frequency Deconvolution Technique by Simulation

This algorithm (found in IRSimulation.py) was developed to test the algorithms described in Sec. B.2.3 that use frequency deconvolution algorithms to obtain the impulse response of the tank environment. This algorithm generates a swept-sine signal as well as a simulated environmental impulse response. The signal and impulse response are convolved together and the resulting simulated recording is processed via the frequency deconvolution algorithm developed. This algorithm may be used with a variety of simple simulated impulse responses that may be generated and simulated in Python. It also offers the ability to add noise to the "recording" in order to evaluate the efficacy of the deconvolution technique shown in Appendix B.2.3. The results of this simulation are discussed in Sec. 3.2.1. Portions of the code are commented out and many parameters may be changed to add more variability to the evaluation.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 26 15:23:29 2021

This python code is designed to simulate a recording and explore processing the
simulated recording via the functions written in ESAUresponse.py
It fist generates a chirped signal. Then it generates one of a few options of
simulated environments (impulse response). These two are convolved with each
other to simulate a recording. The generated chirp and the simulated recording
is passed through SysResponse() from ESAUresponse.py. Noise is also applied
to assess how well it handles under noise at either the input or output (source
or receiver).

Last updated 6/14/2021

```

```
@author: cvongsaw
"""
import ESAUdata as data
import byuarglib as byu
import numpy as np
import ESAUResponse as res
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
params = {'legend.fontsize': 24,
           'figure.figsize': (15, 10),
          'axes.labelsize': 28,
          'axes.titlesize':29,
          'axes.titleweight':'bold',
          'xtick.labelsize':24,
          'ytick.labelsize':24,
          'lines.linewidth':3}
pylab.rcParams.update(params)


"""___SIGNALS____"""
from scipy.signal import chirp
#times at which to evaluate the array for creating the chirp (sig)
chrp0 = 0 #chirp start time (s) (MUST START AT t=0 for CHIRP func)
chrp1 = 0.5 #chirp stop time (s)
f_0 = 10e3 #Hz start freq
f_1 = 100e3 #Hz end freq
fs = 500e3 #sampling rate should be min = 2*f_1
trl0 = 0.1 #trailing & leading zeros
noises = True #compute a noisy signal or not
nLi = 0 #noise Level @ Input/Source (factor, typically 1-10, 10 being VERY noisy)
nLo = 1 #noise Level @ Output/Receive (factor, typically 1-10, 10 being VERY noisy)
tim = np.linspace((chrp0),(chrp1),int(fs*(chrp1-chrp0)))
sig1 = chirp(tim,f_0,chrp1,f_1,method='linear')
#time array for plotting and putting in lead/trail zeros
time = np.linspace(0,(chrp1+2*trl0),int((chrp1+2*trl0)*fs))
nzeros = int(trl0*fs)
sig = np.pad(sig1,(nzeros,nzeros),'constant',constant_values=(0,0))

#divide by convert to change Hz to kHz if 1000 or leave as Hz if 1
if f_0>=1e3:
    convert = 1000
if f_1<=1e3:
    convert = 1

plt.figure()
plt.plot(time,sig)
if convert == 1000:
    plt.title(f'Swept-Sine Signal {f_0/1000}-{f_1/1000} kHz')
if convert == 1:
    plt.title(f'Swept-Sine Signal {f_0}-{f_1} Hz')
plt.xlabel('time (s)')
plt.ylabel('Amplitude')
plt.grid()


"""ADD RANDOM NOISE TO THE SYSTEM"""
if noises == True:
    noise = np.random.normal(0, .1, sig.shape)
    noisy = sig + nLi*noise #simple addition of noise
    plt.figure()
    plt.plot(time,noisy)
    if convert == 1000:
        plt.title(f'Noisy Swept-Sine Signal {f_0/1000}-{f_1/1000} kHz')
    if convert == 1:
        plt.title(f'Noisy Swept-Sine Signal {f_0}-{f_1} Hz')
    plt.xlabel('time (s)')
    plt.ylabel('Amplitude')
    plt.grid()


"""___Simulated Impulse Response___"""
########################################
#arbitrary impulse response for testing#
########################################
from scipy.signal import impulse, unit_impulse
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.impulse.html
system = ([1.0],[3.0,2.0,1.0])
timp,imp = impulse(system)

imp = np.pad(imp,int((len(sig)-len(imp))/2),mode='constant')
################
#delta function#
################
delta = unit_impulse(len(sig),int(0.5*fs))
```

```
#########
#Gausian#
#########
from scipy.signal.windows import gaussian
gauss = gaussian(int(len(time)),std=1)
tg = np.linspace(0,np.max(time),int(len(time)))


"""Impulse Simulation with REFLECTION"""
#append 0.5*imp to imp to simulate a reflection? to be timegated




"""___Plot Impulsive Simulation___"""
"""___This is the target for the Response Code to Return___"""
"""___!!!___CHANGE_INPUTS_HERE_When_Other_IR_Desired___!!!___"""
#time,imp
#delta,time
#gauss,time or tg
RES = imp #IR signal to be tested
t = time #time array for the IR signal to be tested

plt.figure()
plt.plot(t,np.abs(RES))
plt.title('Simulated Impulse Response (IR)')
plt.xlabel('time (s)')
plt.ylabel('Amplitude')
plt.grid()







"""___Convolve Simulated IR w/ Chirp___"""
#import scipy.signal as sci
#sig_flip = np.flip(sig)
#cal1 = sci.correlate(sig_flip,RES,mode='same',method='auto')

cal = np.convolve(sig,RES,mode='same')
```

```
gen = sig

plt.figure()
plt.plot(t,cal)
if convert == 1000:
    plt.title(f'{f_0/1000}-{f_1/1000} kHz Chirp Convolved w/ Simulated IR')
if convert == 1:
    plt.title(f'{f_0}-{f_1} Hz Chirp Convolved w/ Simulated IR')
plt.xlabel('time (s)')
plt.ylabel('Amplitude')
plt.grid()


if noises == True:
    calnoise = np.convolve(noisy,RES,mode='same')
    noise1 = np.random.normal(0, .1, sig.shape)
    calnoise = noise1*nLo + calnoise
    plt.figure()
    plt.plot(t,calnoise)
    if convert == 1000:
        plt.title(f'{f_0/1000}-{f_1/1000} kHz Noisy Chirp Convolved w/ Simulated IR')
    if convert == 1:
        plt.title(f'{f_0}-{f_1} Hz Noisy Chirp Convolved w/ Simulated IR')
    plt.xlabel('time (s)')
    plt.ylabel('Amplitude')
    plt.grid()


"""___Obtain IR & FRF back out___"""

hsys,tsys,Hsys,fsys = res.SysResponse(cal,gen,fs,tgate=0,wiener=True,domain='f')

FRFi = np.fft.fft(RES)
Fi = np.fft.fftfreq(len(RES),d=1/(len(RES)/max(t)))
Fiss = Fi[0:int(len(Fi)/2)]/convert #convert from Hz to kHz
FRFiss = 2*FRFi[0:(int(len(FRFi)/2))]
FRFi_dB = 10*np.log10(np.abs(FRFiss))

"""___Roll the Shape of the Time-Domain___"""
#The IR is shifted to the end of the array, such that the tail
#spills over to the beginning of the ray. The array must be rolled
#for alignment w/ the actual IR. However, the number of zeros must
```

```
#be equal for both leading and trailing zeros.
roll = int(0.5*len(hsys)-1)
hsys = np.roll(hsys,roll)
Hss = 2*Hsys[0:(int(len(Hsys)/2))]   #convert to single-sided FRF
fss = fsys[0:int(len(fsys)/2)]/convert    #convert to single-sided from Hz to kHz
Hss_dB = 10*np.log10(np.abs(Hss))


if noises == True:
    #NOISY VERSION OF SYSTEM RESPONSE
    hsysn,tsysn,Hsysn,fsysn = res.SysResponse(calnoise,gen,fs,tgate=0,wiener=True,
                                              domain='f')
    """___Roll the Shape of the Time-Domain___"""
    #The IR is shifted to the end of the array, such that the tail
    #spills over to the beginning of the ray. The array must be rolled
    #for alignment w/ the actual IR. However, the number of zeros must
    #be equal for both leading and trailing zeros.
    hsysn = np.roll(hsysn,roll)
    Hssn = 2*Hsysn[0:(int(len(Hsysn)/2))]   #convert to single-sided FRF
    fssn = fsysn[0:int(len(fsysn)/2)]/convert#convert to single-sided from Hz>kHz
    Hss_dBn = 10*np.log10(np.abs(Hssn))



"""___PLOT FOR COMPARISON___"""
plt.figure()
plt.plot(t,np.abs(RES),linewidth=6)
plt.plot(tsys,np.abs(hsys),'--',linewidth=3)
if convert == 1000:
    plt.title(f'{f_0/1000}-{f_1/1000} kHz Impulse Response w/ fs={fs/1000}kHz')
if convert == 1:
    plt.title(f'{f_0}-{f_1} Hz Impulse Response w/ fs={fs}Hz')
plt.xlabel('time (s)')
plt.ylabel('Amplitude')
plt.legend(['Simulated IR','Deconvolved IR'])
plt.grid()

if noises == True:
    plt.figure()
    plt.plot(t,np.abs(RES),linewidth=6)
    plt.plot(tsysn,np.abs(hsysn),'--',linewidth=3)
    if convert == 1000:
            plt.title(f'{f_0/1000}-{f_1/1000} kHz Impulse Response w/'
                      +f' Noise & fs={fs/1000}kHz')
        if convert == 1:
            plt.title(f'{f_0}-{f_1} Hz Impulse Response w/ Noise & fs={fs}Hz')
        plt.xlabel('time (s)')
        plt.ylabel('Amplitude')
        plt.legend(['Simulated IR','Deconvolved IR'])
        plt.grid()


"""plt.figure()
plt.plot(Fiss,FRFi_dB,linewidth=6)
plt.plot(fss,Hss_dB,'--',linewidth=3)
if convert == 1000:
    plt.title(f'Frequency Response of {f_0/1000}-{f_1/1000} kHz Signal')
    plt.xlabel('Frequency (kHz)')
if convert == 1:
    plt.title(f'Frequency Response of {f_0}-{f_1} Hz Signal')
    plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude dB')
plt.legend(['Simulated Frequency Response','Deconvolved Frequency Response'])
plt.grid()
buffer_limit = f_1+(f_1-f_0)*0.01
#plt.xlim(f_0-buffer_limit,f_1+buffer_limit)


if noises == True:
    plt.figure()

    plt.plot(fssn,Hss_dBn,'--',linewidth=3,color='tab:orange')
    plt.plot(Fiss,FRFi_dB,linewidth=6,color='tab:blue')
    if convert == 1000:
        plt.title(f'Frequency Response of {f_0/1000}-{f_1/1000} kHz Noisy Signal')
        plt.xlabel('Frequency (kHz)')
    if convert == 1:
        plt.title(f'Frequency Response of {f_0}-{f_1} Hz Noisy Signal')
        plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude dB')
    plt.legend(['Deconvolved Frequency Response','Simulated Frequency Response'])
    plt.grid()
    buffer_limit = f_1+(f_1-f_0)*0.01
    #plt.xlim(f_0-buffer_limit,f_1+buffer_limit)"""
```

```



"""
COULD USE THIS TO TEST OUT T60meas IF I COULD MAKE THE SIGNAL APPEAR REVERBERANT"""
import TankCharacterization as tank
tbound = tank.T60meas_bounds(hsysn,fs)
T60 = tank.T60meas(hsysn,fs,tbound[0],tbound[1],d=0.5,c=1478,rt='T60',plot=True)

"""octData,OctFreq = tank.OctaveFilter(hsysn,f_0,f_1,fs,frac=3)
octTrans = np.transpose(octData)

plt.figure()
for i in range(len(OctFreq)):
    plt.plot(octData[i,:])
plt.title('IR Octave Band')

for i in range(len(OctFreq)):
    tbound = tank.T60meas_bounds(octData[i,:],fs)
    T60 = tank.T60meas(octData[i,:],fs,tbound[0],tbound[1],d=0.5,c=1478,rt='T60',
                        plot=True)
#"""
```

B.3 Tank Characterization

B.3.1 Estimating Characterization Parameters

This algorithm (found in TankCharacterization.py) evaluates an idealized reverberant enclosure using the modified Norris-Eyring equation, as discussed in Sec. 3.1.2, to determine the estimated T_{60}, minimum effective signal length, and estimated Schroeder frequency of the tank.

```


def T60est(d,c = 1478,zi= 3.26e6,ai=0,alpha_p=0):
    """
    Parameters
    ----------
    d:          float;
                depth of water
    c:          float, Optional;
                speed of sound in water. Defaults to 1478m/s (rounded to nearest
                whole value for any depth of water the tank can have), using
                Garrett's eq. for the speed of sound in water relative to
                temperature, depth, and salinity for a temparature of 19 degrees C
                (rough avg. in tank).
    zi:         float, Optional;
                Acoustic impedance of side walls. Defaults to acoustic impdedance
                of acrylic, accepted as 3.26E6 Ns/m**3 from the following source:
                https://www.ndt.net/links/proper.htm
    ai:         float or ndarray of float, Optional;
                Absorption coefficient of tank walls. Defaults to 0 which ignores
                this input. If the absorption coefficient of the walls is known,
                user can input this value and zi will be ignored, solving T60
                using the known absorption. This may also be beneficial when
                accounting for wall anechoic paneling (floor still assumed zi input).
    alpha_p:    float or ndarray of float, Optional;
                Absorption coefficient due to thermoviscous molecular propagation
                losses. Defaults as 0 such that there is no propagation absorption.
                Can use alpha_prop(f,T,S,pH,depth) code to feed in an array of
                frequency dependent absorption coefficients due to propagation
                losses through the water.

    Returns
    -------
    T60:            float;
                    Estimate of the reverberation time (T60) in seconds.
                    i.e. time it takes for the signal to drop by 60dB
    sigL:           float;
                    minimum excitation signal length (s) required by T60 based on
                    Gemba recommendation for 5-10x length of T60. This gives 10x.
    fschroeder:     float;
                    Schroeder Frequency (Hz). The lowest frequency of interest
                    in which the tank is large.


    Notes
    -----
    Author: Cameron Vongsawad

    Comes from Gemba 2014 dissertation ("Characterization of underwater acoustic
    sources recorded in reverberant environments with application to scuba...")
    Section 3.4 equations 3.4.1, 3.4.2, & 3.4.3. Where GembaEQ3.4.1 is the
    Eyring equation that can be found in 661 notes eq 4-2.4.116 calculated from
    the overall estimated Spatially Averaged Absorption Coefficient.

    This can then be used to determine the length of the excitation signal (which
    must be 5-10x longer than the T60)

    This is all relative to the depth of the water, and if the boundary
    impedance is altered from the standard tank. May also input a wall absorption
    coefficient if known. Such as from a measured wall absorption coefficient.
    This currently still assumes the floor absorption is according to zi however.
    This improves all estimations given in this function.

    Can also add in propagation absorption coefficients determined through
    alpha_prop(f,T,S,pH,depth). Or leave that out by allowing the default to
    remain 0. This further improves all estimations given in this function.

    last modified 9/1/2021
    """
    import numpy as np
    #dimensions of tank
    Lx = 1.22   #width of tank (m)
    Ly = 3.66   #length of tank (m)
    V = Lx*Ly*d #volume relative to current water depth
    A_floor = Lx*Ly #total surface area of tank floor
    A_acrylic = A_floor + 2*Ly*d + 2*Lx*d #total surface area of acrylic boundaries
    A_waterair = Lx*Ly #total surface area of water-air boundary
    S = A_acrylic +A_waterair #total surface area of (semi)absorptive boundaries

    #estimate absorption coefficients for boundaries
    zw = 1.5E6 #accepted acoustic impedance of water in Ns/m**3
    za = 415 #accepted acoustic impedance of air in Ns/m**3
    alpha_acrylic = 1-np.abs((zw-zi)/(zw+zi))
    alpha_air = 1-np.abs((zw-za)/(zw+za))
    Aw = alpha_air*A_waterair/S #water absorption coefficient spatially averaged

    if ai == 0:
        #using zi (estimated acoustic impedance of walls)
        #Sum of alpha*A/S found in eq. 3.4.1 of Gemba
        #Absorption can be more thoroughly estimated using Physcs 661 notes.
        Ai = alpha_acrylic*(A_acrylic)/S #acrylic absorp coeff spatially averaged
        Absorb = Aw + Ai #spatially averaged Absorption Coefficient
    else:
        #using ai (estimated acoustic absorption coefficient of walls)
        Awall = ai*(A_acrylic-A_floor)/S
        Ai = alpha_acrylic*(A_floor)/S #acrylic absorp coeff spatially averaged
        Absorb = Aw + Ai + Awall #spatially averaged Absorption Coefficient

    #Eyring equation (661 notes eq. 4-24.124(reduce to 4-2.4.116 when alpha_p=0))
    T60 = (24*np.log(10)/c) * (V/(8*alpha_p*V - S*np.log(1-Absorb)))
    fschroeder = np.sqrt(c**3*T60/(V*4*np.log(10))) #Pierce eq6.6.4
    signal_length = 10*T60
    sigL = signal_length

    #if desired to compare with a simpler room estimation found in 461 notes?
    #T60ng = np.log(10**6)*4*V/(c*Absorb)

    return T60, sigL, fschroeder
```

B.3.2 Signal and Recording Length

The following algorithm follows Muller-Trapet [36] calculation for the minimum trailing zeros that must be included in a signal to allow a recording to observe efficient decay of a swept-sine signal and is found in TankCharacterization.py.

```
def trailzeros(RT,sigL,fstart,fstop,f = None,R = 60,sig = 'lin') :
    """
    *****Super not sure if this is working because linear and exponential dont
    give differing results. and tstop does not seem like it is calculated
    correctly since when f = None should cause it to give 1 as solution********


    Parameters
    ----------
    RT:     float;
            Reverberation time (s). Defaults to the T60, but can be altered by
            changing the following parameter R. T60 estimate can be determined
            by the depth of the water using the function T60est
    sigL:   float;
            Generated Signal length (s). The min. length can be found in
            T60est(d) and should be 5-10x that of the estimated T60
    fstart: float;
            Start frequency (Hz) of the chirp signal
    fstop:  float;
            Stop frequency (Hz) of the chirp signal
    f:      float;
            Target Frequency of interest within the chirped signal, often the
            highest frequency and therefore defaults as None which makes
            f = fstop. Chosen as the highest frequency of interest.
    R:      float, Optional;
            Defaults to 60dB as the dynamic range for the reverberation time
            (T60), but can be change to a T15, T25, etc.
    sig:    string, Optional;
            Signal type. Either 'lin' for linear or 'exp' exponential chirp.
            Defaults to 'lin' chirped signal.

    Returns
    -------
    tstop:  float;
            Trailing zeros necessary (stop margin, or stop gap)
    tls:    float;
            Total length of signal and trailing zeros recommended.

    Notes
    -----
    Author: Cameron Vongsawad

    Trailing Zeros estimate from Muller-Trapet JASA 2020 paper based on RT.

    Changed order of input to align better with T60est()

    Last Modified: 2/22/2021

    """
    #tf = time in the sweep, when certain frequency f is played
    #D = dynamic range for RT found by D = 20dB + R where R is the reverb time
    # level decrease (where for a T60 will be R = 60 and D = 80)
    import numpy as np

    if f == None:
            f = fstop
    if sig == 'lin' or 'linear':
        #eq 21 time in the sweep when the target frequency occurs (sigL = min.
        #actual sweep len)
        tlin = sigL*(f-fstart)/(fstop-fstart)
        #eq 20 determining the total signal/recording length duration from
        #dynamic range and the estimated RT
        l = tlin + (20 + R)/60*RT
        #eq 18 determine the stop margine or time of trailing zeros for the
        #signal (l = total signal duration, sigL = time of sweep design)
        tstop = l - sigL

    if sig == 'exp' or 'exponential':
        #eq 24 time in the sweep when the target frequency occurs (sigL = min.
        #actual sweep len) for exponential chirps
        texp = sigL* np.log(f/fstart)/np.log(fstop/fstart)
        #eq 20 determining the total signal/recording length duration from
        #dynamic range and the estimated RT
        l = texp + (20 + R)/60*RT
        #eq 25 determine the stop margine or time of trailing zeros for the
        #signal (l = total signal duration, sigL = time of sweep design)
        tstop = ((20+R)/60*texp*np.log(fstop/fstart) \
                 - l *np.log(fstop/f))/np.log(f/fstart)

    tls = l
    print(sigL)
    print(tstop)
    print(tls)
    return tstop, tls
```

B.3.3 Propagation Absorption

Acoustic propagation absorption through water is determined following the formulation of Ainslie and McColm [45] to determine frequency and range-dependent absorption of acoustic energy through water. The Ainslie and McColm model outputs the solution of the model in dB/km and should be converted to Np/m for effective processing in accordance with the code used in Appendix B.3.6. The code for this algorithm is found in TankCharacterization.py.

```python
def alpha_prop(f,T=16,S=5,pH=7.7,depth=0.6):
    """
    Absorption Coefficient from Propagation losses through Sea Water

    Parameters
    ----------
    f:      ndarray of float;
            frequency array for bandwidth of interest for freq. dependent absorption
    T:      float, Optional;
            Temperature of the water in Celcius. Defaults to 16 degrees C.
            Effective for -6<T<35 degrees C.
    S:      float, Optional;
            Salinity of the water in ppt. Effective for 5<S<50 ppt. Defaults to
            S = 5 ppt
    pH:     float, Optional;
            pH level of the water. Defaults to 7.7 (though this is high relative
            to the test strips and a normal pool). Effective for 7.7<pH<8.3
    depth:  float, Optional;
            depth of water in km. Defaults to 0.6m or 0.0006 km. Which will
            make that term in the function negligible as basically zero.
            Effective for 0<z<7000m or 0<z<7 km.

    Returns
    -------
    a_p:    ndarray of float;
            absorption coefficient if sound (alpha) for propagation
            losses through the water. (Np/m or Nepers/m)

    Notes
    -----
    Author: Cameron Vongsawad

    Primarily due to viscous effects above 100kHz (high), but also due to
    Chemical relaxation of Boric Acid up to a few kHz (low), Chemical
    relaxation of Magnesium Sulfate up to a few 100kHz (mid). This formulation
    comes from Ainslie & McColm 1997 - "A simplified formula for viscous and
    chemical absorption in sea water" Published in JASA 103 equation 2 & 3.

    Can apply this in propagation models similar to account for thermoviscous
    molecular losses.

    For reference: http://resource.npl.co.uk/acoustics/techguides/seaabsorption/

    last modified 9/1/2021
    """
    import numpy as np
    #The function originally takes in km, This converts m (used in lab) to km
    depth = depth/1000
    #relaxation frequency for boron
    f1 = 0.78*(S/35)**0.5*np.exp(T/26)
    #relaxation frequency for magnesium
    f2 = 42*np.exp(T/17)

    term1 = 0.106*(f1*f**2)/(f**2+f1**2)*np.exp((pH-8)/0.56)
    term2 = 0.52*(1+T/43)*(S/35)*(f2*f**2)/(f**2+f2**2)*np.exp(-depth/6)
    term3 = 0.00049*f**2*np.exp(-(T/27 + depth/17))
    a_p = (term1 + term2 + term3)
    #Original function returns solution in dB/km
    #convert dB/km to Np/m (Nepers/meter)
    a_p = a_p/1000 *0.115129254650564
    return a_p
```

B.3.4 Time Bounds for Reverse Schroeder Intergration

This algorithm (found in TankCharacterization.py) generates a GUI in order to evaluate and determine good time bounds to perform reverse Schroeder integration on an impulse response squared. This evaluation of the time bounds follows the standards for determining the T_{60} through reverse Schroeder integration according to ISO 354:2003, ISO3382-1:2009, and ISO3382-2:2008.

```
#before the following function can be used, currently the variable below must
#be initialized to ensure it will function due to a conditional statement used.
l1 = None #DO NOT ERASE
def T60meas_bounds(data,fs):
    """
    Parameters
    ----------
    data:       Ndarray;
                Impulse Response data.
    fs:         Float;
                Sampling rate.

    Returns
    -------
    tbounds:    List (2 values);
                List of the initial and final time bounds to perform the
                reverse Schroeder integration on T60meas(data,fs,t0,t1,d,c,rt,plot)

    Notes
    -----
    Author: Cameron Vongsawad

    Utilize a pop up graph to view the 10log10(h(t)**2) of an input impulse
    response or h(t). This allows you to choose the appropriate time bounds
    to perform the reverse Schroeder integration on the impulse response h(t)
    according to ISO354:2003, ISO3382-1:2009, ISO3382-2:2008 standards.

    *****Because of the conditional statement in updatePlot(), the statement:
    "l1=None" must remain before this function. This simply initializes l1
    until another workaround is determined.

    This is to be passed into the T60meas code in order to plot the decay curve
    and determine the T60 of the impulse response. OctFilter() is recommended
    prior to this function in order to pass through specific frequency bands.
    When doing this, it is also recommended that you write a loop to loop
    through each octave band through this function.

    last modified 7/19/2021
    """


    #Creatre fonts to be used in the plotting.
    LARGE_FONT= ("Verdana", 12)
    #Medium_FONT= ("Verdana", 10)

    #import necessary packages for use in GUI and plotting
    import tkinter
    from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
    from matplotlib.backends.backend_tkagg import NavigationToolbar2Tk
    # Implement the default Matplotlib key bindings.
    from matplotlib.backend_bases import key_press_handler
    from matplotlib.figure import Figure

    #Setup the main window for the GUI
    root = tkinter.Tk()
    root.wm_title("Check h(t)**2")
    label = tkinter.Label(root, text="Impulse Response Squared in dB",
                          font=LARGE_FONT)
    label.pack(pady=10,padx=10)
    #Create a figure to plot
    import numpy as np
    fig = Figure(figsize=(10, 8), dpi=100)
    ax = fig.add_subplot(111)#.plot(t,data)
    #Create time array for the sample data (IR)
    t = np.linspace(0,len(data)/fs,len(data))
    #Grab data to plot and adjust to properly view h(t)**2
    floor = np.max(data)*1e-5
    ht = np.clip(data,floor,(np.max(data)+floor)) #data w/out zeros
    b = 10*np.log10((np.abs(ht))**2)
    #Plot initial data
    fig.suptitle("Choose a Time Interval (t1 to t2) for Reverse Schroeder"
                 +"Integration. This should be where the plot is most linear.")
    ax.set_xlabel("Time (s)")
    ax.set_ylabel("Level (dB)")
    ax.plot(t,b)
    ax.grid(True)
    fig.canvas.draw_idle()

    #Create inputs for choosing bounds on the graph
    in1_label = tkinter.Label(root,text="t1",font=LARGE_FONT)
    root.in1 = tkinter.Entry(root)
    in2_label = tkinter.Label(root,text="t2",font=LARGE_FONT)
    root.in2 = tkinter.Entry(root)
    in1_label.pack(side="left", fill="x", expand = False)
    in2_label.pack(side="right", fill="x", expand = False)
    root.in1.pack(side="left", fill="x", expand = False)
    root.in2.pack(side="right", fill="x", expand = False)

    #Replace vlines for checking bounds on the 10log10(h(t)**2) plot
    #when the Check button is pressed.
    def updatePlot(t1,t2):
        tt1 = float(t1)
        tt2 = float(t2)
        global l1
        global l2
        if l1 != None:
            ax.lines.remove(l1)
            ax.lines.remove(l2)
            fig.canvas.draw_idle()
        l1 = ax.axvline(tt1,color="orange",linestyle="--")
        l2 = ax.axvline(tt2,color="orange",linestyle="--")
        fig.canvas.draw_idle()

    #calculate the reverberation time based on the time bounds chosen when
    #Run T60mean button is pressed.
    def Reverb(t1,t2):
        tt1 = float(t1)
        tt2 = float(t2)
        global tbounds
        global t60
        tbounds = [tt1,tt2]
        root.destroy()

    #A tk.DrawingArea.
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()

    #Create a toolbar for the GUI including ability to save plot.
    toolbar = NavigationToolbar2Tk(canvas, root)
    toolbar.update()
    canvas.mpl_connect("key_press_event", lambda event: print(
            f"you pressed {event.key}"))
    canvas.mpl_connect("key_press_event", key_press_handler)

    #Create buttons for control in GUI
    run_button = tkinter.Button(root,text="Run T60meas",command=lambda: Reverb(
            root.in1.get(),root.in2.get()))
    check_button = tkinter.Button(root,text="Check",command=lambda: updatePlot(
            root.in1.get(),root.in2.get()))

    # Packing order for Widgets are processed sequentially.
    # The canvas is rather flexible in its size, so we pack it last which makes
    # sure the UI controls are displayed as long as possible.
    run_button.pack(side="bottom")
    check_button.pack(side="bottom")
    toolbar.pack(side=tkinter.BOTTOM, fill="y")#tkinter.X)
    canvas.get_tk_widget().pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=1)
    #Loop GUI
    tkinter.mainloop()

    #Return the bounds in a list and the T60 time.
    return tbounds
```

B.3.5 Measured Reverberation Time

This algorithm (found in TankCharacterization.py) determined the T_{60} from an input impulse response and time bounds determined from the algorithm discussed in Sec. B.3.4.

```
def T60meas(ht,fs,t0,t1,d=0.6,c=1478,rt='T60',plot=False):
    """
    Calculate the T20, T30, or T60 from Backward Schroeder Integration on the
    measured impulse response hsys.

    Parameters
    ----------
    ht:         ndarray of float;
                Measured Impulse Response of the environment.
    fs:         float;
                Sampling frequency of the impulse reponse.
    t0:         int;
                start time in seconds
    t1:         int;
                finish time in seconds
    d:          float, Optional;
                depth of water. Defaults to a common 0.6m of water in the tank
    c:          float, Optional;
                speed of sound in water. Defaults to 1478 rounded to nearest whole
                value for any depth of water the tank can have, using Garrett's eq.
                for the speed of sound in water relative to temperature, depth, and
                salinity for a temparature of 19 degrees C (rough avg. in tank).
    rt:         String, Optional;
                Choose desired Reverb Time (rt) as T10, T20, T30, or T60. Defaults
```

```
                    to T60. Choosing less than T60 estimates the T60 by assuming linear
                    relationship between chosen rt and T60.
    plot:           boolian, Optional;
                    Defaults to False so as to not Plot the 10log(h(t)**2) and the
                    associated Decay Curve. True would plot the two.


    Returns
    -------
    T60:            float;
                    Calculated reverberation time (T60) in the tank in seconds.
                    This is calculated using the Through The System (TTS)
                    response to evaluate reverberation only in the tank.
                    (i.e. time it takes for the signal in the tank to drop by
                    60dB)

    Notes
    -----
    Author: Cameron Vongsawad

    Calculate the measured T60 in the tank.

    Some guidance for this part found here:
    https://github.com/python-acoustics/python-acoustics/blob/master/acoustics/room.py
    the above link provides an alternate method to more generalize this solution

    This also follows ISO3382-1:2009(E)

    last modified 5/18/2021
    """
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.pylab as pylab
    from scipy import stats
    params = {'legend.fontsize': 24,
              'figure.figsize': (15, 10),
              'axes.labelsize': 28,
              'axes.titlesize':29,
              'axes.titleweight':'bold',
              'xtick.labelsize':24,
              'ytick.labelsize':24,
              'lines.linewidth':3}
    pylab.rcParams.update(params)

    ##avoid log(0) to prevent exploding by clipping all zeros to 0.00001% of
    ##the max and go just beyond the max value so as to not clip.
    floor = np.max(ht)*1e-5
    ht1 = np.clip(ht,floor,(np.max(ht)+floor)) #data w/out zeros

    #Portion of array to actually look at for the T60meas. This is found by
    #eyeing it (ISO3382-1:2009(E)). Look just after the 10*np.log10(np.abs(ht)**2)
    #is flat in the beg. and just before it is flat in the end (background level).
    t0 = int(t0*fs) #convert to samples
    t1 = int(t1*fs) #convert to samples
    ht1 = ht1[t0:t1]

    #Backward Schroeder Integration
    T = 1/fs
    schroeder = np.cumsum(ht1[::-1]**2)[::-1]*T
    schroeder_dB = 10*np.log10(schroeder)

    if rt == 'T10':
        #determine T10 between -5dB and -15dB of the max value of the decay curve
        init = -5.0
        end = -15.0
        factor = 6.0 #amount to mult. T10 by to extrapolate T60
    if rt == 'T20':
        #determine T20 between -5dB and -25dB of the max value of the decay curve
        init = -5.0
        end = -25.0
        factor = 3.0 #amount to mult. T20 by to extrapolate T60
    if rt == 'T30':
        #determine T30 between -5dB and -35dB of the max value of the decay curve
        init = -5.0
        end = -35.0
        factor = 2.0 #amount to mult. T30 by to extrapolate T60
    if rt == 'T60':
        #determine T60 between -5dB and -65dB of the max value of the decay curve
        init = -5.0
        end = -65.0
        factor = 1.0 #amount to mult. T60 by to extrapolate T60

    #Relative value to refine search for init & end bounds for rt measurement.
    maxval = np.max(schroeder_dB)
    schroeder_dB = schroeder_dB - maxval
    #Linear regression
    #determine the value on the decay curve where it is nearest the init and end
    #values below the maximum of the the decay curve
    sch_init = schroeder_dB[np.abs(schroeder_dB - init).argmin()]
    sch_end = schroeder_dB[np.abs(schroeder_dB - end).argmin()]

    check_actual = (sch_init - sch_end +5)
    check_bounds = (init-end)
    if check_actual < check_bounds:
        raise ValueError(f"Decay not large enough for {rt} measurement."
                         +"Choose smaller rt value.")

    #indices of where the decay curve matches the init and end condition
    init_sample = np.where(schroeder_dB == sch_init)[0][0]
    end_sample = np.where(schroeder_dB == sch_end)[0][0]

    #Reverberation time (RT)
    #convert samples to time and determine the difference
    t_init = init_sample / fs
    t_end = end_sample / fs
    RT = t_end - t_init
    T60 = factor*RT
    print('T60 =',T60,'s')
    print('')
    print('')


    if plot == True:
        t = np.linspace(0,len(ht1)/fs,len(ht1))
        Level = 10*np.log10((np.abs(ht1))**2)
        plt.figure()
        #plot the IR**2 in dB
        plt.plot(t,Level)
        plt.xlabel('Time (s)')
        plt.ylabel('Level (dB)')
        plt.grid()
        #plot Decay Curve
        plt.plot(t,(schroeder_dB + maxval))
        plt.legend([r'$10log[h^{2}(t)]$','Decay Curve'])
        est,_,_ = T60est(d,c)
        plt.title(f'T60meas={np.around(T60*1000,decimals=2)}ms,'
                                        +f'T60est={np.around(est*1000,decimals=2)}ms')

    return T60
```

B.3.6 Boundary Absorption

The following algorithm uses the measured T_{60} to determine the spatially averaged absorption of the tank boundaries. The user may choose to account for the absorption at the water-air surface and/or absorption due to propagation. This algorithm is found in TankCharacterization.py.

```
def alpha_wall(T60,d=0.6,c=1478,acc=False,alpha_p=0):
    """
    Calculate the spatially averaged absorption coefficient of the walls of the
    tank based on the measured T60 of the tank (either averaged or over freq)

    Parameters
    ----------
    T60:        float or array of float;
                Calculated reverberation time (T60) in the tank in seconds.
                This is calculated using the T60meas function.
    d:          float, Optional;
                depth of water. Defaults to a common 0.6m of water in the tank
    c:          float, Optional;
                speed of sound in water. Defaults to 1478 rounded to nearest whole
                value for any depth of water the tank can have, using Garrett's eq.
                for the speed of sound in water relative to temperature, depth, and
                salinity for a temparature of 19 degrees C (rough avg. in tank).
    acc:        boolian, Optional;
                Account for the assumption that the water-air boundary is perfectly
                reflective. Defaults to False to not make this assumption and give
                the overall spatially averaged absorption coefficient. If True,
                then the spatially averaged absorption coefficient that is returned
                only accounts for the walls and the floor of the enclosure.
    alpha_p:    float or ndarray of float, Optional;
                Absorption coefficient due to thermoviscous molecular propagation
                losses. Defaults as 0 such that there is no propagation absorption.
                Can use alpha_prop(f,T,S,pH,depth) code to feed in an array of
                frequency dependent absorption coefficients due to propagation
```

```
                  losses through the water.

    Returns
    -------
    alpha_S:      float;
                  Estimated spatially averaged absorption coefficient
                  (Nepers/m**2) for the room based on the measured T60 and the
                  Norris-Eyring Equation.

    Notes
    -----
    Author: Cameron Vongsawad

    Calculate the spatially averaged absorption coefficient of the tank boudaries
    from the measured T60 in the tank.

    This assumes that the incident energy per unit time and area is the same for
    all wall surfaces at any given time. If one of the consituent areas S1 has
    an absorption coefficient alpha1 that is substantially different than the
    coefficient alpha0 of the remaining surface area S-S1, this assumption
    becomes questionable. See 661 notes section 4-2.4.2.4 (Rooms with
    asymmetric or nonuniform absorption).


    last modified 10/21/2021
    """
    import numpy as np
    #dimensions of tank
    Lx = 1.22   #width of tank (m)
    Ly = 3.66   #length of tank (m)
    V = Lx*Ly*d #volume relative to current water depth
    S = 2*(Lx*Ly+Ly*d+d*Lx) #total enclosed surface area including air-water

    if acc == True:
        #account for perfectly reflective water-air boundary
        print('Calc. absorp. assuming pressure-release water-air (reflective)')
        Aw = 0
    else:
        print('Calc. absorption w/ respect to small surface absorption')
        #account for slight impedance water-air boundary
        #estimate absorption coefficients for boundaries
        zw = 1.5E6 #accepted acoustic impedance of water in Ns/m**3
        za = 415 #accepted acoustic impedance of air in Ns/m**3
        alpha_air = 1-np.abs((zw-za)/(zw+za))
        Aw = alpha_air*Lx*Ly/(S) #water absorption coefficient spatially averaged

    #solving Eyring equation w/propagation loss found in 661 notes eq 4-2.4.124
    #with default of alpha_p=0 this simplifies to eq. 4-2.4.116.
    alpha_S = 1-np.exp(V/S*(8*alpha_p - (24*np.log(10))/(c*T60)))

    #Account for the absorption of the water-air surface boundary while
    #assuming that the incident energy per unit time and area is the same for
    #all wall surfaces at any given time. (This does not apply asymmetric correction)
    A = alpha_S*S - Aw
    alpha_S = A/S
    return alpha_S
```

B.3.7 Added Absorption

When absorptive material is placed within the tank environment, the absorption of the added material may be determined from this algorithm (found in TankCharacterization.py) by comparing the measured T_{60} with and without the material added.

```
def alpha_addition(ai,T60_1,T60_2,dS,d,c=1478):
    """
    Determine the absorption coefficient of any added material in the tank by
    comparison of a pre and post T60 measurement. This is performed by solving
    for the change in absorptive area A between two measurements using the
    Sabine equation. (Solving this for the Eyring Eq. may be a better idea)

    Parameters
    ----------
    ai:         float;
                Measured or estimated absorption of tank acrylic walls that are
                being covered by input material
    T60_1:      float;
                Measured reverbeation time in seconds of the tank prior to
                application of new material into the tank.
    T60_2:      float;
                Measured reverbeation time in seconds of the tank post
```

```
                 application of new material into the tank.
    dS:          float;
                 Effective change in surface area (m**2) of the tank boundaries
                 due to application of new material.
    d:           float;
                 Depth of the water in m
    c:           float, Optional;
                 Speed of sound in m/s. Defaults to 1478 m/s

    Returns:
    alpha_add:   float;
                 Measured absorption (Nepers) of material input into the tank
                 based on reverberation time prior to placing new material.

    """
    import numpy as np
    #dimensions of tank
    Lx = 1.22   #width of tank (m)
    Ly = 3.66   #length of tank (m)
    V = Lx*Ly*d #volume relative to current water depth
    #661 eq 4-2.4.90 & 4-2.4.91 generalized for any sound speed
    alpha_add = ai+24*np.log(10)*V/(c*dS)*(1/T60_2 - 1/T60_1)
    return alpha_add
```

B.4 Models

B.4.1 Eigenmodes

This algorithm determines the eigenmodes and eigenfrequencies of an idealized rigid wall tank solution. The rigid wall solution with pressure-release water-air surface may be used for the reasons discussed in Sec. 3.3.3. This algorithm is found in TankCharacterization.py.

```
def TankMode(perm=10,fmin=0,fmax=1000,Lx=1.22,Ly=3.66,Lz=0.6,c=1478):
    """
    Determine the rigid wall condition Eigenmodes, Eigenfrequencies, and
    Eigenfunctions for any frequency range and a chosen number of permutations

    Parameters
    ----------
    perm:   float, Optional;
            number of permutations to iterate through for each dimension x, y, z.
            Will end up with arrays for f and mode of size perm**3. Default is 10.
    fmin:   float, Optional;
            Minimum frequency of interest to reduce computation time and limit
            output array size. Defaults to 0 Hz
    fmax:   float, Optional;
            Maximum frequency of interest to reduce computation time and limit
            output array size. Defaults to 1000 Hz
    Lx:     float, Optional;
            Width of water tank. Defalts as 1.22 m for the BYU Underwater tank.
            This could be altered when anechoic panels are placed in the tank.
    Ly:     float, Optional;
            Length of water tank. Defalts as 3.66 m for the BYU Underwater tank.
            This could be altered when anechoic panels are placed in the tank.
    Lz:     float, Optional;
            Depth of water in the tank. Defalts as 0.6 m for the BYU Underwater
            tank. This SHOULD be altered dependent on the current water level
            in the tank.
    c:      float, Optional;
            Speed of sound in water. This defaults to 1478 m/s following Garrett's
            formula for speed of sound due to Depth, Salinity, and Temperature.
            This Default is set to the average room temperature of the water and
            assuming near zero salinity over any depth the tank can handle.


    Returns
    -------
    f:      ndarray of float;
            Ordered Natural frequencies of the tank environment assuming Rigid
            walls and a pressure release surface.
    mode:   ndarray of int;
            Associated mode numbers of the natural frequencies of the tank.

    Notes
    -----
    Author: Cameron Vongsawad

    Calculate the natural frequencies and room modes as defined by Garrett
    eq. 13.12 altered by kz in 13.14 for pressure release boundary
    (aka solutions to the eigenfrequencies for a rigid walled tank with pressure
    release water-air interface)

    last modified 4/7/2021
    """

    import numpy as np
    #Perfectly rigid wall solution
    print('Solving for natural frequencies assuming perfectly Rigid walls')
    #rigid wall solution for natural frequencies &
    #Pressure release surface(nz component)
    fN = lambda nx,ny,nz: c/(2)*np.sqrt((nx/Lx)**2 + (ny/Ly)**2 + (
            (2*nz-1)/(2*Lz))**2)
    #create empty lists to populate
    mode = []
    f = []
    #iterate through the permutations selected for each mode possibility nx,ny,nz
    #nx,ny: 0,1,2,3...
    for nx in range(0,perm):
        for ny in range(0,perm):
            #nz: 1,2,3...
            #Garrett pg.721 "The nz = 0 solution does not exist since constant
            #pressure in the z-direction is not an option that satisfies the
            #boundary conditions at z=Lz and z=0 simultaneously."
            for nz in range(1,perm):
                #for only values within the chosen bandwidth fmin<= f <=fmax
                temp = fN(nx,ny,nz)
                if temp >=fmin:
                    if temp <= fmax:
                        f.append(fN(nx,ny,nz))
                        mode.append([nx,ny,nz])

    f = np.array(f)
    mode = np.array(mode)
    #order all the frequencies & associated modes in numerical order of freq.
    idxs = np.argsort(f)
    f = f[idxs]
    mode = mode[idxs]
    print(f'{len(f)} frequencies recorded in range {fmin}<=f<={fmax}')
    return f, mode
```

B.4.2 Eigenfunctions

The eigenfunctions corresponding to the above eigenmodes for a rigid wall solution are determined in this algorithm found in TankCharacterization.py. The rigid wall solution with pressure-release water-air surface may be used for the reasons discussed in Sec. 3.3.3.

```

def TankFunc(x,y,z,f,mode,Lx=1.22,Ly=3.66,Lz=0.6,plot=True,pstyle='colored',num=4):
    """
    Parameters
    ----------
    x:      float or array;
            Single value for a single position in the tank, or array of values
            to iterate over.
    y:      float or array;
            Single value for a single position in the tank, or array of values
            to iterate over.
    z:      float or array;
            Single value for a single position in the tank, or array of values
            to iterate over.
    f:      Ndarray of float;
            Ordered Eigenfrequency array calculated and output by TankMode function
    mode:   Ndarray of float;
            Ordered Eigenmode array calculated and output by TankMode function
    Lx:     float, Optional;
            Width of water tank. Defalts as 1.22 m for the BYU Underwater tank.
            This could be altered when anechoic panels are placed in the tank.
    Ly:     float, Optional;
            Length of water tank. Defalts as 3.66 m for the BYU Underwater tank.
            This could be altered when anechoic panels are placed in the tank.
    Lz:     float, Optional;
            Depth of water in the tank. Defalts as 0.6 m for the BYU Underwater
            tank. This SHOULD be altered dependent on the current water level
            in the tank.
    plot:   Boolian; Optional;
            Choose whether or not to plot the EigenFunctions of the natural
            frequencies in the x-y, x-z, and y-z planes for the first "num" of
            modes. Default is set as True to plot. False will not plot. This only
            plots if len(x) or len(y) or len(z) != 1
    pstyle: string; Optional;
```

```
            Defaults to 'colored' to plot contourf plots. Can also choose 'line'
            to plot contour line plots. The latter is only recommended when solving
            for very low frequencies.
    num:    float; Optional;
            Number of modes to plot if Plot = True. Default is set to 4 modes for
            each coordinate plane for a total of 12 plots (if is shows anything)

    Returns
    -------
    psi:    3D ndarray of float;
            Ordered Eigenfunctions of the natural frequencies.

    Notes
    -----
    Author: Cameron Vongsawad

    Calculate the natural frequencies and room modes as defined by Garrett
    eq. 13.12 altered by kz in 13.14 for pressure release boundary
    (aka solutions to the eigenfrequencies for a rigid walled tank with pressure
    release water-air interface)

    last modified 4/7/2021
    """
    import numpy as np
    #Eigen-Function for rectangular tank assuming rigid walls and pressure release
    #water-air interface
    Psi = lambda nx,ny,nz :np.cos(nx*np.pi*x/Lx) * np.cos(ny*np.pi*y/Ly) * np.cos(
            (2*nz-1)*np.pi*z/(2*Lz))

    psi = []
    print('')
    print('calculating EigenFunctions')
    for i in range(len(mode)):
            psi.append(Psi(mode[i,0],mode[i,1],mode[i,2]))
    psi = np.array(psi)


    if len(x) > 1:
        ##########################################
        #The rest of this function is solely for #
        #contour plotting the Eigenfunctions psi #
        ##########################################
        if plot == True:
            print(f'plotting first {num} EigenFunctions')
            import matplotlib.pyplot as plt
            import matplotlib.pylab as pylab
            params = {'legend.fontsize': 24,
                      'figure.figsize': (15, 10),
                     'axes.labelsize': 28,
                     'axes.titlesize':29,
                     'axes.titleweight':'bold',
                     'xtick.labelsize':24,
                     'ytick.labelsize':24,
                     'lines.linewidth':3}
            pylab.rcParams.update(params)
            #number of modes we are interested in contour plotting
            start = 0   #zeroeth mode 0, 0, 0 is weird?.
            #modes and frequencies of interest
            modeint = mode[start:num+start]
            fint = f[start:num+start]

            #plot over x-y plane w/ z = 0
            #create spatial arrays for the 3-dimensions of the tank
            x = np.linspace(0,Lx)
            y = np.linspace(0,Ly)
            z = 0
            x,y = np.meshgrid(x,y)
            for i in range(len(modeint)):
                psi1 = Psi(modeint[i,0],modeint[i,1],modeint[i,2])
                #check if mode actually present in this plane, if not, do not plot
                check =np.ones((len(x),len(y)))
                if np.any(psi1 != check) == True:
                    fig,ax=plt.subplots(1,1)
                    if pstyle == 'line':
                        cb = ax.contour(x,y,psi1,colors='black',linestyles='dashed')
                        ax.clabel(cb,inline=True,fontsize=15)
                    else:
                        cb = ax.contourf(x,y,psi1)
                        fig.colorbar(cb)
                    ax.set_title(f'{modeint[i,:]} Mode f={np.round(fint[i],2)} Hz'
                                + f'where Z={z}m')
                    ax.set_xlabel('X (m)')
                    ax.set_ylabel('Y (m)')
                    plt.show()
                else:
                    print('undesired mode not plotted in x-y')

            #plot over x-z plane w/ y = 0
            #create spatial arrays for the 3-dimensions of the tank
            x = np.linspace(0,Lx)
            z = np.linspace(0,Lz)
            y = 0
            x,z = np.meshgrid(x,z)
            for i in range(len(modeint)):
                #iterate through calculating each eigenfunction
                psi2 = Psi(modeint[i,0],modeint[i,1],modeint[i,2])
                #check if mode actually present in this plane, if not, do not plot
                check =np.ones((len(x),len(z)))
                if np.any(psi2 != check) == True:
                    fig,ax=plt.subplots(1,1)
                    if pstyle == 'line':
                        cb = ax.contour(x,z,psi2,colors='black',linestyles='dashed')
                        ax.clabel(cb,inline=True,fontsize=15)
                    else:
                        cb = ax.contourf(x,z,psi2)
                        fig.colorbar(cb)
                    ax.set_title(f'{modeint[i,:]} Mode f={np.round(fint[i],2)} Hz'
                                    + f'where Y={y}m')
                    ax.set_xlabel('X (m)')
                    ax.set_ylabel('Z (m)')
                    plt.show()
                else:
                    print('undesired mode not plotted in x-z')

            #plot over y-z plane w/ x = 0
            #create spatial arrays for the 3-dimensions of the tank
            y = np.linspace(0,Ly)
            z = np.linspace(0,Lz)
            x = 0
            y,z = np.meshgrid(y,z)
            for i in range(len(modeint)):
                #iterate through calculating each eigenfunction
                psi3 = Psi(modeint[i,0],modeint[i,1],modeint[i,2])
                #check if mode actually present in this plane, if not, do not plot
                check =np.ones((len(y),len(z)))
                if np.any(psi3 != check) == True:
                    fig,ax=plt.subplots(1,1)
                    if pstyle == 'line':
                        cb = ax.contour(y,z,psi3,colors='black',linestyles='dashed')
                        ax.clabel(cb,inline=True,fontsize=15)
                    else:
                        cb = ax.contourf(y,z,psi3)
                        fig.colorbar(cb)
                    ax.set_title(f'{modeint[i,:]} Mode f={np.round(fint[i],2)}'
                                    + f'Hz where X={x}m')
                    ax.set_xlabel('Y (m)')
                    ax.set_ylabel('Z (m)')
                    plt.show()
                else:
                    print('undesired mode not plotted in y-z')
    else:
        #not sure what this plot physically means, but I have it here for now.
        #might need to change default to not plotting if singular value is input.
        if plot == True:
            import matplotlib.pyplot as plt
            import matplotlib.pylab as pylab
            params = {'legend.fontsize': 24,
                      'figure.figsize': (15, 10),
                     'axes.labelsize': 28,
                     'axes.titlesize':29,
                     'axes.titleweight':'bold',
                     'xtick.labelsize':24,
                     'ytick.labelsize':24,
                     'lines.linewidth':3}
            pylab.rcParams.update(params)
            plt.figure()
            plt.plot(f,psi)
            plt.xlabel('Frequency (Hz)')
            plt.ylabel(r'$\Psi (r)$')
            plt.title(rf'Eigenfunction $\Psi$({x},{y},{z})')

    return psi
```

B.4.3 Finite-Impedance Model

This algorithm (found in Tank Characterization.py) follows the solution discussed in Sec. 3.3.3 following the Pierce [43] solution for a finite-impedance boundary water tank.

```

def P_model_Pierce(Psi0,Psi,k,kn,mode,alpha,d=0.6,A=1,acc=False,anech=False,
                   alpha_p=0):
    """
    Parameters
    ----------
    Psi0:       Ndarray of float;
                Eigenfunctions at source position. Determined w/ TankFunc function.
    Psi:        Ndarray of float;
                Eigenfunctions at receiver position. Determined w/ TankFunc function.
    k:          Ndarray of float;
                Wavenumber of frequency band of interest.
    kn:         Ndarray of float;
                Eigenmodes of the tank environment. Determined w/ TankMode function
    mode:       Ndarray of float;
                Ordered Eigenmode array calculated and output by TankMode function
    alpha:      Ndarray of float;
                Spatially averaged absorption coefficient
    d:          float, Optional;
                depth of water. Defaults to a common 0.6m of water in the tank
    A:          float, Optional;
                Amplitude of the function. A=1 (default) ensures the solution is
                simply the Green's function.
    acc:        boolian, Optional;
                Account for the assumption that the water-air boundary is perfectly
                reflective. Defaults to False to not make this assumption and give
                the overall spatially averaged absorption coefficient. If True,
                then the spatially averaged absorption coefficient that is returned
                only accounts for the walls and the floor of the enclosure.
    anech:      boolian, Optional;
                Defaults to False to calculate with no anechoic panels in the tank.
                If true, the calculation takes into account the thickness of the
                panels on the inner dimensions of the tank environment for the
                calculation of the spatially averaged absorption coefficient.
    alpha_p:    float or ndarray of float, Optional;
                Absorption coefficient due to thermoviscous molecular propagation
                losses. Defaults as 0 such that there is no propagation absorption.
                Can use alpha_prop(f,T,S,pH,depth) code to feed in an array of
                frequency dependent absorption coefficients due to propagation
                losses through the water.

    Returns
    -------
    P:          Ndarray of float;
                Green's function or pressure in the tank as a function of source
                and receiver positions.

    Notes
    -----
    Author: Cameron Vongsawad

    This follows Pierce "Acoustics: An Introduction to its Physical Principles
    and Applications" 3rd Ed. eq 6.5.20 (2019) or more precisely Leishman 661
    notes 4-2C eq. 4-2.2.193 (2021)

    last modified 5/19/2021
    """
    import numpy as np
    #dimensions of tank
    Lx = 1.22   #width of tank (m)
    Ly = 3.66   #length of tank (m)
    #Alter the dimensions relative to the thickness of the anechoic panels
    if anech == True:
        Lx = Lx - 2*0.05
        Ly = Ly - 2*0.05
        print('Calculating w/ respect to anechoic panels')
    V = Lx*Ly*d #volume relative to current water depth
    if acc == True:
        S = 2*(Ly*d+d*Lx)+Lx*Ly #total enclosed surface area minus air-water surface
        print('Calculating w/ respect to walls only')
    else:
        S = 2*(Lx*Ly+Ly*d+d*Lx) #total enclosed surface area including air-water
        print('Calculating w/ respect to water surface & walls')
    #Spatially averaged absorption area including propagation absorption
    alpha_wall = alpha #wall absorption/impedance accounted for
    As = S*alpha_wall + 8*alpha_p*V
    x,y,z = mode[0],mode[1],mode[2]
    if x == 0:
        Ex = 1
    else:
        Ex = 2
    if y == 0:
        Ey = 1
    else:
        Ey = 2
    Ez = 2
    lamb = 1/(Ex*Ey*Ez)
    P= -4*np.pi*A*np.sum((Psi*Psi0)/(V*lamb*(k**2-kn**2-1j*k*(As/(4*V)))))

    return P
```

B.5 Processing Data: Putting the Algorithms Together

The following code comes from TestCode.py which pulls from each algorithm discussed above in order to process the measured data to obtain the reverberation time T_{60} and spatially averaged absorption coefficient $\langle\alpha(f)\rangle_S$. The results of this computation are saved in .xlsx spreadsheets discussed Appendix A.3. Adaptations to TestCode.py were made in TestCode2.py (not copied here because it is so similar to TestCode.py) in order to determine the integration time bounds (see Sec. 4.3) for the reverse Schroeder integration [49, 50] from randomly selected measurements. TestCode5.py (not shown here) is basic code for reading .xlsx files and plotting the reverberation and absorption data for analysis. TestCode5.py is not shown simply because it does not contain much other than calling data and plotting commands but may be of interest for reference.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jul  6 12:21:13 2020

TEST CODE USED FOR PROCESSING ALL OF THE DATA FROM THE LARGE SCAN
THIS COLLECTS THE T60 and Absorption OF THE FULL SCAN

@author: cvong
```

```python
"""

import numpy as np
#import byuarglib as byu
#import sys as directory
#keep using the natural working directory
#directory.path.insert(0,'C:/Users/cvongsaw/Box/UW Research/Code/uw-measurements')
#add a second working directory
"""directory.path.insert(1,'C:/Users/cvongsaw/Box/UW Research/Code/
                        underwater-measurements/analysis/')#"""
from readLogFile import readLogFile
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
params = {'legend.fontsize': 24,
          'figure.figsize': (15, 10),
         'axes.labelsize': 28,
         'axes.titlesize':29,
         'axes.titleweight':'bold',
         'xtick.labelsize':24,
         'ytick.labelsize':24,
         'lines.linewidth':3}
pylab.rcParams.update(params)
import scipy.signal as sci
import ESAUpose as pose
import ESAUdata as data
import ESAUResponse as response
import TankCharacterization as tank
import TimeGate_UnderwaterTank as tg
import xlsxwriter as xls
import openpyxl as xl
#import pdb
#pdb.set_trace()

################################################################################
################################################################################
#when switching file name via copy/paste, must change forward and back slashes
################################################################################
################################################################################
date4 = '2021-02-24' #scan 3,7,8(noise),9(noise)
date5 = '2021-02-26' #TL measurement [scan 3(5 points) and 4(10 points) good]
date6 = '2021-03-23' #short cal length long signal length
date11 = '2021-08-09'
```

```
date12 = 'Need to redo date12 good tests with panels in'
date13 = '2021-09-13'
date14 = '2021-09-14'
date15 = '2021-09-15'
date16 = '2021-09-16'
date17 = '2021-09-17B'
date18 = '2021-09-22'
date19 = '2021-09-27'
date2 = date15 #second file name
date = date13 #first file name

###NO PANELS####
#scan 2, 5k-10k 2021-09-14 downsample false [0.511,0.5155] octs 30
#scan7, 10k-50k 2021-09-15 downsample true factor2 [0.511,0.5149] octs 30
#scan 9, 50k-100k (24:729) 2021-09-15 [0.5103,0.514]
#scan 11 50k-100k (0:24)  [0.5103,0.514] 2021-09-16 downsample factor2? octs 30
#scan 1, 100k-500k (1V) 2021-09-17B\2021-09-17 octs 25?
#scan 2, 100k-500k (3V) 2021-09-17B\2021-09-17 octs 25 (0:10)[0.6004,0.606]
###With PANELS####
#scan 3, 5k-10k 2021-09-27 octs 30 [0.6008,0.604]
#scan 4, 10k-50k 2021-09-27 octs 30 [0.6005,0.6025]
#scan 6, 50k-100k 2021-09-27 octs 30 [0.6005,0.6031]
#scan 1, 100k-500k (1V) 2021-09-22 octs 25 ?
#scan 4, 100k-500k (3V) 2021-09-22 octs 25 [0.6005,0.6035]
scan = '9'
octs = 30 #int(len(OctFreq)) #number of octave bands to look at
tbound = [0.5103,0.514]
group = np.arange(716,729,1) #set of desired measurement IDs
downsample = False #may need to adjust factor
factor = 4 #factor by which to downsample
bandwidth = '50k-100k'
propagation = True #account for propagation absorption losses
water_air = False #account for minute impedance boundary of water-air surface
wall1 = "AbsorbAcrylic" #input data into appropriate spreadsheet name
wall2 = "AbsorbPanels" #input data into appropriate spreadsheet name
wall3 = "AbsorbAcrylicProp" #input data into appropriate spreadsheet name
wall4 = "AbsorbPanelsProp" #input data into appropriate spreadsheet name
walltype = wall3

for iii in group:
    print(f'Scan {iii} Calculation in progress...')
    #desire is the list of all scans you care to actually look at in this analysis
```

```python
    desire = [iii]#,19,56,80]
    filename = f'ID{iii}_000log.txt'
    calfile = 'cal_000log.txt'
    channels = [0,1] #recording channels of interest
    #recorded calibration of interest (currently should only be len(cal_channel)=1)
    #legend = ['0']
    cal_channel = [1]
    #legend = ['Near Wall','Middle','Anechoic']

    year = date[0:4]
    xls_file = f'W:/Vongsawad/data/{date}/{walltype}{bandwidth}.xlsx'
    path = f'W:/Vongsawad/data/{date}/{date2[0:10]}_scan{scan}/'

    freqMin,freqMax,temp,fs,startzero,sigL,trail,totdur,depth,xSource,ySource,\
    zSource,xRec,yRec,zRec = readLogFile(filename,path)
    _,_,_,_,_,_,_,_,_,xA_cal,yA_cal,zA_cal,xR_cal,yR_cal,zR_cal = readLogFile(
            calfile,path)
    temp = 20.88
    depth = 0.5

    N = fs*sigL #number of samples
    signal = f'{sigL}s Chirp {freqMin/1000}kHz-{freqMax/1000}kHz'
    test = f'{date} scan{scan} {signal}'

    #generated CALIBRATION signal
    fscal = fs #sampling frequency
    startzerocal = 0.02 #leading zeros of the signal
    sigLcal = sigL #signal length
    trailcal = 0.5 #trailing zeros
    treccal = sigLcal #time record length in sec
    Ncal = fscal*treccal
    Acal = (xA_cal,yA_cal,zA_cal)#(0.6,2.14,depth/2)
    Rcal = (xR_cal,yR_cal,zR_cal)#(0.6,2.06,depth/2)

    ##############################################################################
    ##############################################################################
    ##############################################################################
    ##############################################################################
    #load generated signal and calibration measurement
    """gen, calgen, cal, ch0, ch1, ch2, ch3 = data.ESAUdata(path+scan, desire,
                                                         channels, N, Ncal)#"""

    gen, _, cal, ch0, ch1, _, _ = data.ESAUdata(path, desire, channels, N, Ncal)
    calgen=gen

    #Need to downsample the signal for the T60 code to remove extra noise to
    #IR**2 to at most 2.5x fmax
    if downsample == True:
        #resample the data for higher sampling rates???
        factor = factor #factor by which to reduce signal
        gen = sci.decimate(gen,factor)
        calgen = sci.decimate(calgen,factor)
        ch0 = sci.decimate(ch0[:,0],factor)
        ch1 = sci.decimate(ch1[:,0],factor)
        fs = fs/factor #"""
        fscal = fs #sampling frequency
        caldec = np.empty([len(ch0),4])
        for i in range(4):
            caldec[:,i] = sci.decimate(cal[:,i],factor)
        cal = caldec

    #Load in positions, calculate range distance, plot scan positions desired
    A,R,dd = pose.ESAUpose(path, desire, plot=False, Acal = Acal, Rcal = Rcal)


    #time delays now allowing for a single measurement w/ single source/receiver pose
    c = tg.uwsoundspeed(D=depth,T=temp,S=0.03, model='Garrett')


    #need to update MeasGreen to handle various channels.and update the inputs
    #to use ch number instead of allsignals.
    print('')
    print('')
    print('System Response...')
    #obtain only the cal of interest for calculating the system response
    cal1 = np.ndarray.flatten(cal[:,cal_channel])
    #cal1 = cal[:,cal_channel[0]] ###!!change!!!####
    #cal1 = ch0[:,0]
    #ch1 = ch1[:,desire]
    #ch1 = ch1[:,0] ##!!change!!!####

    tgate,_,tdir,dis = tg.gateValue(A[0],R[0],c)
    tgatec,_,tdirc,disc = tg.gateValue(Acal,Rcal,c)
```

```python
    #how much to gate the signal in samples
    tb4gate = 0.1 #ms before first reflection tgate
    Nb4gate = tb4gate/1000 *fs #convert to samples before gating
    Ngate = tgate*fs-Nb4gate

    tt = np.linspace(0,len(cal1)/fs,len(cal1))      #time array for recorded signal
    tt = tt*1000
    """plt.figure()
    plt.plot(tt,ch1)
    plt.title(f'Recorded Signal \n {signal} fs={fs/1000}kHz \n {date} scan{scan}')
    plt.xlabel('time (ms)')
    plt.ylabel('Amplitude')"""

    hsys,tsys,Hsys,fsys = response.SysResponse(cal1,calgen,fscal,tgate=tgate,
                                                wiener=True,domain='f')
    if downsample == False:
        Htank,ftank = response.TankResponse(ch1[:,0],gen,fs,hsys,wiener=True,
                                            domain='f')
    if downsample == True:
        Htank,ftank = response.TankResponse(ch1,gen,fs,hsys,wiener=True,domain='f')
    htank = np.real(np.fft.ifft(Htank))
    ttank = np.linspace(0,len(htank)/fs,len(htank))


    """Hss = 2*Hsys[0:(int(len(Hsys)/2))]   #convert to single-sided FRF
    fss = fsys[0:int(len(fsys)/2)]/1000    #convert to single-sided from Hz to kHz
    Hss_dB = 10*np.log10(np.abs(Hss))   #convert to Levels (dB)

    plt.figure()
    plt.plot(fss,Hss_dB)
    plt.title(f'FRF of {signal} for System')
    plt.xlabel('Frequency (kHz)')
    plt.ylabel('Amplitude')
    plt.grid()

    plt.figure()
    plt.plot(tsys*1000,hsys)
    plt.axvline(tgatec*1000,linestyle='dashed',color='g')
    #no idea how to gate this relative to fs
    #plt.axvline(Ngate*1000,linestyle='dashdot',color='r')
    #need to compare this with the fs = 1M and fs = 10M measurements.
    plt.axvline(tdirc*1000,linestyle='dashed',color='r')
    plt.xlabel('Time (ms)')
    plt.ylabel('Amplitude')
    plt.legend(['Chirp IR','Estimated First Reflection','Estimated Direct Signal'])
    plt.title(f'Time-Gated Calibration IR of {signal} fs ={fs}Hz')
    plt.xlim(0,0.8)
    plt.grid()"""





    """___Roll the Shape of the Time-Domain for htank___"""
    #The IR is shifted to the end of the array, such that the tail
    #spills over to the beginning of the ray. The array must be rolled
    #for alignment w/ the actual IR. However, the number of zeros must
    #be equal for both leading and trailing zeros.
    roll = True
    if roll == True:
        rollt = int(0.5*len(htank)-1)
        shift = rollt/fs
        htank = np.roll(htank,rollt)
    else:
        shift = 0
    """Htss = 2*Htank[0:(int(len(Htank)/2))]   #convert to single-sided FRF
    ftss = ftank[0:int(len(ftank)/2)]/1000    #convert to single-sided from Hz to kHz
    Htss_dB = 10*np.log10(np.abs(Htss))   #convert to Levels (dB)

    plt.figure()
    plt.plot(ftss,Htss_dB)
    plt.title(f'FRF of {signal} for Tank')
    plt.xlabel('Frequency (kHz)')
    plt.ylabel('Amplitude')
    plt.grid()
    #"""

    #####################################################
    """plt.figure()
    plt.plot(ttank,htank)
    plt.axvline(tgate+shift,linestyle='dashed',color='g')
    #no idea how to gate this relative to fs
    #plt.axvline(Ngate*1000,linestyle='dashdot',color='r')
    #need to compare this with the fs = 1M and fs = 10M measurements.
    plt.axvline(tdir+shift,linestyle='dashed',color='r')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.legend(['Chirp IR','Estimated First Reflection','Estimated Direct Signal'])
    plt.title(f'Calibrated IR of {signal} fs ={fs}Hz')
    #plt.xlim(0,3)
    plt.grid()"""


    #tbound = tank.T60meas_bounds(htank,fs)
    T60 = tank.T60meas(htank,fs,tbound[0],tbound[1],d=depth,c=c,rt='T10',plot=False)

    octData,OctFreq = tank.OctaveFilter(htank,freqMin,freqMax,fs,frac=octs)
    octTrans = np.transpose(octData)

    if propagation == True:
        #propagation absorption estimated for the water characteristics over desired
        #Octave Bands prop=0 #when desired to not look into propagation effects.
        prop = tank.alpha_prop(OctFreq,T=temp,S=5,pH=7.2,depth=depth)
    else: prop = np.zeros(len(OctFreq))

    """plt.figure()
    legend1 = []
    for i in range(len(OctFreq)):
        plt.plot(ttank,octData[i,:])
        legend1.append(np.round(OctFreq[i]))
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.title('IR Octave Band')
    plt.legend(legend1)"""

    """plt.figure()
    legend = []
    for i in range(len(OctFreq)):
        plt.plot(ttank,octTrans[:,i])
        legend.append(np.round(OctFreq[i]))
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.title('IR Octave Band decimated')
    plt.legend(legend1)"""


    T60_f = np.empty(int(len(OctFreq)))
    a_wall_f = np.empty(int(len(OctFreq)))
    for i in range(len(OctFreq)):
        #tbound = tank.T60meas_bounds(octTrans[:,i],fs)
        T60_f[i] = tank.T60meas(octTrans[:,i],fs,tbound[0],tbound[1],d=depth,c=c,
             rt='T10',plot=False)
        #plt.suptitle(f'f = {OctFreq[i]/1000} kHz')
        a_wall_f[i] = tank.alpha_wall(T60_f[i],d=depth,c=c,acc=water_air,
                alpha_p=prop[i])

    #overall absorption coefficient over entire bandwidth. Cannot include
    #propagation absortion which is freq. dependent for a full scan bandwidth
    a_wall_gen = tank.alpha_wall(T60,d=depth,c=c,acc=water_air,alpha_p=0)
    T60est,sig,fschroeder = tank.T60est(depth,c=c,)



    #Open an excel file and append data to it to create a repository of data
    #this is particularly important for sharing absorption coefficient values
    #to apply to the models Kaylyn is working on.
    #https://realpython.com/openpyxl-excel-spreadsheets-python/
    #https://openpyxl.readthedocs.io/en/stable/tutorial.html
    #https://www.geeksforgeeks.org/python-writing-excel-file-using-openpyxl-module/
    wb = xl.load_workbook(xls_file)     #Find created worksheet
    a_sheet = wb['Alpha']               #Find worksheet names
    T_sheet = wb['T60']
    #populate selected excel workbook with alpha (absorption coefficient data)
    for i in range(len(OctFreq)):
        c1 = a_sheet.cell(row=1,column=i+3)
        c1.value = OctFreq[i]               #populate alpha frequencies
        c1 = T_sheet.cell(row=1,column=i+3)
        c1.value = OctFreq[i]               #populate T60 frequencies
    for i in range(len(a_wall_f)):
        c01 = a_sheet.cell(row=desire[0]+3,column = 2)
        c01.value = a_wall_gen              #populate overall alpha
        c2 = a_sheet.cell(row=desire[0]+3,column=1)
        c2.value = f'alpha_{desire[0]}'     #order alpha values by ID#
        c3 = a_sheet.cell(row=desire[0]+3,column=i+3)
        c3.value = a_wall_f[i]              #populate alpha values by freq bin

    #populate selected excel workbook with T60 (Reverberation time data)
    for i in range(len(a_wall_f)):
        c01 = T_sheet.cell(row=desire[0]+3,column = 2)
        c01.value = T60                     #populate overall T60 value
        c2 = T_sheet.cell(row=desire[0]+3,column=1)
        c2.value = f'T60_{desire[0]}'       #order T60 values by ID#
        c3 = T_sheet.cell(row=desire[0]+3,column=i+3)
        c3.value = T60_f[i]                 #populate T60 values by freq bin
    wb.save(xls_file)    #save the updated workbook



"""
ht = np.abs(np.fft.ifft(Htank))
T60 = tank.T60meas(ht,fs,t0=0,t1=int(len(ht)/fs),d=depth,c=c,rt='T60',plot=True)
alpha_s = tank.alpha_wall(T60,depth,c,acc=False,anech=False,alpha_p=0)
"""
"""
Hss = 2*Hsys[0:(int(len(Hsys)/2))]   #convert to single-sided FRF
fss = fsys[0:int(len(fsys)/2)]/1000    #convert to single-sided from Hz to kHz
Hss_dB = 10*np.log10(Hss)

Hsstank = 2*Htank[0:(int(len(Htank)/2))]   #convert to single-sided FRF
fsstank = ftank[0:int(len(ftank)/2)]/1000    #convert to single-sided from Hz to kHz
Hsstank_dB = 10*np.log10(Hsstank/1e-6)
"""
"""
ir = response.IR(ch1,gen,fs,wiener=False,domain='f')
ir1 = 10*np.log10(ir**2)
tt = np.linspace(0,len(ir1)/fs,len(ir1))               #time array for ht
tt = tt*1000
frf = np.fft.fft(ir)
frf = 10*np.log10(frf/1e-6)
frf = 2*frf[0:(int(len(frf)/2))]
f = np.fft.fftfreq(len(ir),d=1/fs)
f = f[0:int(len(f)/2)]/1000
plt.figure()
plt.plot(tt,ir)
plt.title('straight IR')
plt.xlabel('time (ms)')
plt.ylabel('')
plt.grid()
plt.figure()
plt.plot(f,frf)
```

```
plt.title(f'Straight FRF')
plt.xlabel('Frequency (kHz)')
plt.ylabel(r'Level (dB re 1 $\mu Pa$)')
plt.grid()
"""




"""
plt.figure()
plt.plot(fss,Hss_dB)
plt.title(f'Time-Gated Calibration FRF of {signal} Swept-Sine')
plt.xlabel('Frequency (kHz)')
plt.ylabel(r'Level (dB re 1 $\mu Pa$)')
#plt.xlim(0,300)
#plt.ylim(35,65)
plt.grid()

plt.figure()
plt.plot(fsstank,Hsstank_dB)
plt.title(f'Calibrated Tank Transfer Function of {signal} Swept-Sine')
plt.xlabel('Frequency (kHz)')
plt.ylabel(r'Level (dB re 1 $\mu Pa$)')
#plt.xlim(0,300)
#plt.ylim(35,65)
plt.grid()
"""







"""
#need to EDIT with the below link
#https://www.askpython.com/python/examples/rmse-root-mean-square-error

"""


```

```
##############################################################################
##############################################################################
#OASPL w/ and w/out panels
##############################################################################
##############################################################################

##############################################################################
"""OASPL"""
##############################################################################
"""
#Calculate the overall sound pressure level with respect to 1 microPascal as is
#standard for underwater acoustics, rounded to 4 decimal places.
print('')
print('calculating OASPL re 1e-6 Pa from timewaveform...')
OASPL = np.empty(len(desire))
for i in range(len(desire)):
    OASPL[i] = np.round(10 * np.log10( np.mean( np.square( ch0[:,i] ) ) /1e-6**2),4)
print(f'OASPL{legend} = {OASPL}')
"""
##############################################################################
"""
gate_dir = np.argwhere(t<0.34e-3)
gate_dir = max(gate_dir[:,0])
gate_min = np.argwhere(t>0.34e-3)
gate_min = min(gate_min[:,0])
gate_max = np.argwhere(t<0.40e-3)
gate_max = max(gate_max[:,0])
IR_gate_dir = np.real(x0[:gate_dir, 0])
IR_gate_anech = np.real(x0[gate_min:gate_max, 2])
IR_gate_acryl = np.real(x0[gate_min:gate_max, 0])

#OASPL
OASPL = np.empty((1,len(desire)))
for i in range(len(desire)):
    OASPL[:,i] = byu.OASPLcalc(ch0[:,i])
print(OASPL)
"""



##############################################################################
##############################################################################
```

```

"""
#Calculate the overall sound pressure level with respect to 1 microPascal as is
#standard for underwater acoustics, rounded to 4 decimal places. This is
#calculating from the IR instead of the timewaveform as seen above.
print('')
print('calculating IR OASPL re 1e-6 Pa...')
OASPL = np.empty(len(desire))
for i in range(len(desire)):
    OASPL[i] = np.round(10 * np.log10( np.mean( np.square( x0[:,i] ) ) /1e-6**2),4)
print(f'OASPL{legend} = {OASPL}')
"""

```

Appendix C

Laboratory Documentation

A Guide to Equipment, Maintenance, and Measurements in the Hydroacoustics Lab

This is an active laboratory document developed to help answer frequently asked questions and concerns that may be encountered within the BYU underwater acoustics research laboratory. This is the first version of this lab documentation and constitutes a detailing of the tank, measurement chain, and general equipment in the laboratory. This documentation offers a guide to taking measurements, maintaining equipment, and cleaning the tank environment. This document is key to the safe and effective operation of experimentation within the BYU underwater acoustic research laboratory located in U117 of the Eyring Science Center at Brigham Young University.

C.1 Water Tank

The 1.22 m wide by 3.66 m long by 0.91 m open-air water tank was made by Engineering Laboratory Design Inc. and is made of scratch resistant acrylic. However, it can and will likely become scratched over time and care should be given to maintain the clear and smooth nature of the acrylic.

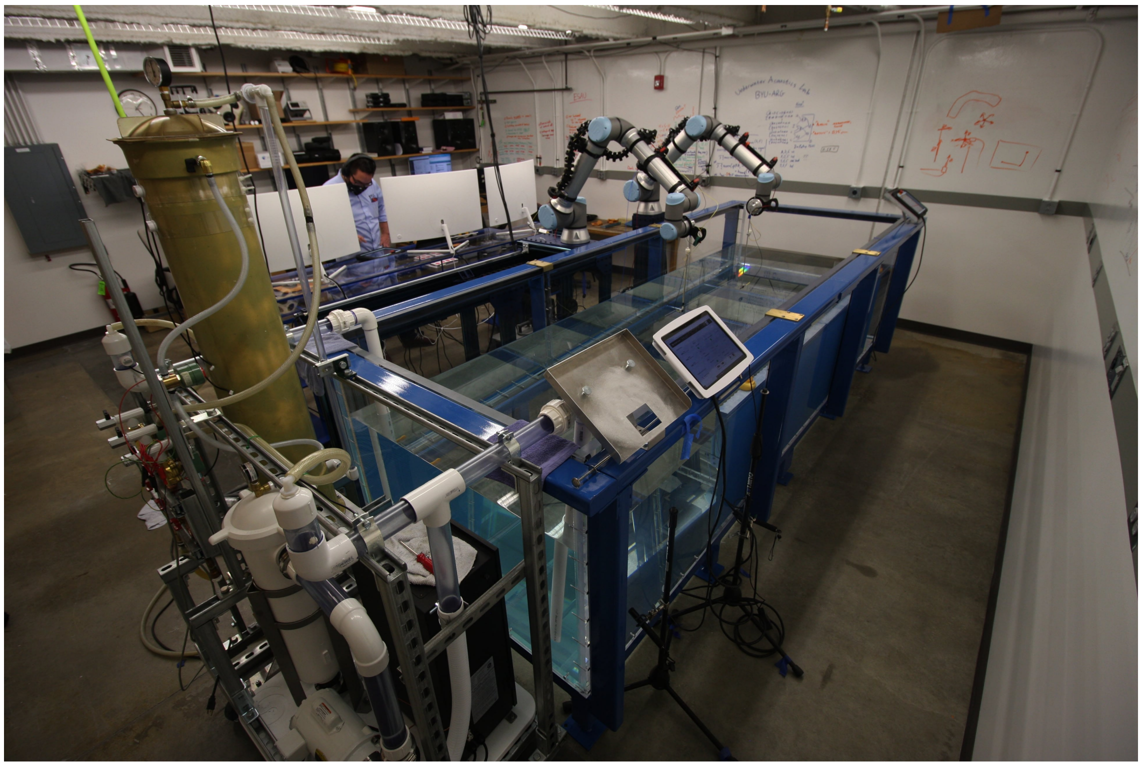


Figure C.1 Hydroacoustics Lab at BYU's Eyring Science Center

The acrylic should acoustic properties similar to the following characteristics: speed of sound 2750 m/s, density 1.19g/cm^3, acoustic impedance 3.26, and attenuation 6.4 dB/cm at 5MHz. Exact physical characteristics could be measured using an 11"x11" sample of the acrylic obtained from Engineering Laboratory Design Inc.. Initial measurements determined the sample to have an approximate density of 1.175g/cm^3 by mass and dimensions. Acoustic characteristics of the tank may also be measured via in-situ measurements within the tank. For more information on how to characterize the tank, see Cameron Vongsawad's masters thesis 2021.

C.1.1 Maintenance

To avoid electrical issues related to water, all maintenance of the tank should be done on the West side of the tank (Xmax side wall) between the teach pendants. This will ensure that potential spills or drips do not affect the robot, tablets or any of the other measurement equipment. All maintenance should be recorded in the “Tank Maintenance.xls” spreadsheet found on the Atlantis Computer Desktop.

Always keep a detailed record of any type of maintenance performed on the tank, whether cleaning, filtering, draining, filling, fixing, etc. This should be saved in the “Tank Maintenance.xls” spreadsheet found on the Atlantis Computer Desktop.

A link to app.poolcalculator.com can be found here as well as all the details necessary to be filled in to the poolcalculator. Simply fill in the day’s date and the associated important details and you will be informed what to fill into the app.

Supplies needed: Filtration system, Clorox pool test strips, Safety goggles, Nitrile gloves, Bleach(8.25%), Hydrochloric Acid(34.6% Muriatic Acid), Baking soda, Distilled water, Graduated cylinder, Funnel, Microfiber towels (grey). Water treatment should be done about once a month, the filtration unit should be able to handle the bulk of keeping the water clean with its particle filter and UV cleaning treatment. More supplies may be purchased with our account at the Chem Store in the Nicholes Building directly South of the Eyring Science Center. You must obtain a slip from the Chem Store to be signed by Traci in order to be added as a purchaser on the account (this will require your name, netID and the lab rm #(U117esc)).

Checking Tank Levels

Tank levels should be checked once every other week. Using app.poolcalculator.com in association with the Clorox test strips above the sink.

- Settings should be selected for Type: Pool, Surface: Fiberglass, Unit Type: Metric, Ranges: Traditional Pool
  - Volume = (3.66m x 1.22m x depth x1000) L. This can be calculated by using the website’s Volume Calculator in the side menu.

  - Temperature = (Reading on Duck, thermocouples, or ESAU) in Celsius
  - Input the tested and goal marks for the Chlorine, pH, Alkalinity and Calcium Hardness to be instructed on what to adjust automatically.

- Clorox test strips
  - Dip the test strip into the water to about elbow depth and remove immediately. (If water is too low for elbow depth, get at least half-way under the water and hold for about the time it would take to go to your elbow and then remove, too long can over-saturate the test strip).
  - Compare colors as best as you can to those on the bottle
  - Record findings in the online pool calculator and the maintenance record.

Water Treatment

- Always wear safety goggles and gloves when handling chemicals. Goggles and gloves can be found in the cabinet above the sink. A facemask may also be beneficial particularly when handling HCL.
- Bleach (used for chlorinating the tank)
  - Ensure to put concentration of bleach in the pool calculator to obtain proper amount needed to add (should be 8.25%)
  - Fill graduated cylinder with appropriate amount of bleach, dilute with distilled water and carefully submerge the graduated cylinder and run it along the length of the tank from (X,0,Z) to (X,Y,Z) gradually allowing the diluted bleach to pour out into the length of the tank without creating bubbles. Gently rinse with tank water carefully to not splash or create bubbles.

- HCI (used for lowering pH) *Consider wearing a face mask when handling the HCL.
  - Ensure to put concentration of HCL in the pool calculator to obtain the proper amount needed to add (should be 36.25-38
  - Fill graduated cylinder with appropriate amount of HCL, dilute with distilled water. Carefully submerge the entire graduated cylinder close to the water surface and move it along the length of the tank from (X,0,Z) to (X,Y,Z) gradually allowing the diluted HCL to spread out into the length of the tank without creating bubbles and evenly spreading it. Gently rinse the graduated cylinder with tank water carefully to not splash or create bubbles.
- Washing Soda, Borax, or Soda Ash (used for raising the alkalinity and pH of the water.)
  - Fill the graduated cylinder with the appropriate amount of chosen compound according to the pool calculator.
  - Gently spread the compound out along the length of the tank.
  - Gently mix in the compound using the pool skimmer until it is mostly dissolved into the water. Follow this by turning on the filtration system for at least an hour.

Filtration

The filtration unit designed and built by John Ellsworth has particle filtration, UV treatment, a water heating element, and debubbling. Filtration should be performed for 4 hours per day to maintain cleanliness.

- Ensure the water pressure is appropriate in the filtration system as indicated by the red LED on the top of the intake and output. If the pressure is not appropriate, the water level on the intake and output will be low and the LED will not be lit. If the LED is not lit, the pump will not activate until air is purged from the system and the water level rises so that the LEDs are lit.

- The filtration system is fitted with quick release inlet and outlet piping. Once the pressure is released from both the inlet and the outlet, these may be removed to remove the filtration system from the tank. When these are reattached, both the inlet and outlet must be re-pressurized on the control panel until the LED in either pipe is restored with the water level about an inch above the LED water level sensor.
- Filtration is not enough on its own and the water chlorine and pH levels should still be checked regularly.

Filling and Draining the Tank

- Filling
  - The tank should be initially filled with tap water from the main faucet. The tank should never be filled with only distilled water because of the corrosive nature of a nutrient/mineral deficient water. Distilled water can easily eat metal up in an attempt to obtain minerals.
  - Attach the clean hose (neon colored) to the right faucet at the sink for tap water
  - Attach the clean hose to the left faucet at the sink for distilled water. Distilled water may be used once the tank is at least half full (0.4m) if less hard water is desired. However, it should mainly be used for refilling a tank that has lost water level due to evaporation, but maintains the original total hardness.
  - Ensure the clean hose (neon colored) is attached to the right side of the hose splitter under the tank. Should be the side not already attached to drain.

  - Turn on the water while the tank valve is still closed but the drain valve is open. This will purge any impurities that may remain in the line. This can be tricky (for distilled water) because it will not stay on by itself without forcing the valve to stay open somehow (use of a claw clamp and tripod is helpful).
  - Close the drain valve on the splitter. Then remove the magnetic plug in the bottom of the tank and open the valve to allow water to flow into the tank.
  - When filled to desired depth, close the valve on the tank, allow excess water to drain down the dirty hose line, and disconnect the clean hose from the faucet. Leave both splitter valves open to ensure the hose can dry out properly.

- Draining
  - In the case of needing to drain, a magnetic rod must first be used to remove the plexiglass plug in the base.
  - The greywater line (dirty) is denoted by the grey hose attached to the splitter and permanently in the floor drain.
  - Simply turn the splitter valve on (toward the center) to allow flow as well as open the large yellow handle valve to the tank to drain. Cover both ends with old rags to catch drips
  - Clean the squeegee with alcohol to prevent any debree from scratching the acrylic surfaces when used. Then use the squeegee to ensure all water drains as best as possible.

- Cleaning the acrylic tank walls
  - First drain the tank.
  - Prepare a small and clean bucket filled with water and a drop or few of dish soap ready with a few clean gray microfiber towels.

- Pour a little bit of the lightly soapy water down the drain to clean out the last bit as much as possible and wash it through the hose and down the drain.
- Use towels and lightly soapy water to wipe down and then dry all acrylic surfaces to remove all hard water, chemical, dust or dirt residue in the tank. Wring out towels in a different bucket than the clean soapy water.
- Take care to not scratch the acrylic. Wear clean socks and gloves when entering the tank and put most weight over the steel supports. A mask may also be worn in order to prevent any impurities attaching to the tank walls/floor.

C.2 Underwater Positioning System

The Underwater Positioning System (UPS) uses two UR10e collaborative robots from Universal-Robots.

Ægir, named after the Norse god of the sea, was obtained first in order to evaluate if it met the need of our specific applications. Ægir (**Host name:** [redacted] **Static IP address:** [redacted]) originally came with version 5.5 of the polyscope software. Ægir is mounted on a simple stationary Vention pedestal on the South-East end of the water tank. Rán (**Host name:** [redacted] **Static IP address:** [redacted]) is Ægir's wife in Norse mythology and the second UR10e robot obtained that originally came with version 5.6 of the polyscope software. Rán is mounted on a Vention MachineMotion (**Host name:** [redacted] **Static IP address:** [redacted]) 7th-axis extender track (https://vention.io/designs/ur-7th-axis-with-extended-range-of-20ft-with-urcaps-integration-64710) that allows maximum reach throughout the water tank environment. Each robot currently runs on URsoftware **5.7.0.90932** software updated on January 21, 2020. (some details have been redacted for security purposes)

Since the UR10e robots are collaborative, force sensors and other safety features make it safe and intuitive for students to use. The polyscope robot software is simple to program with the help of a basic online training course found on universal-robots.com/academy that also provides a thorough understanding of setting up robot safety protocols. The Core, Advanced, Application and Interface training should be completed before attempting to alter any robot programming, manually move the robots, or adapt any ESAU code for robot motion control. It is also recommended to meet with an already trained student after completing the online training to practice using the robots with someone to answer any questions that may come up.

Positioning with these robots is precise within 0.01mm. The added Vention MachineMotion 7th axis extender adds an additional ±0.01mm of uncertainty primarily seen at the far end of the track. Care should be given to program the proper offset positioning of any tool end connector for the robot arm.

Custom end connectors have been designed and built in house for holding each specific transducers used in the underwater acoustic measurement tank. Care has been taken for both ensuring transducer safety and optimizing orientation for transducer directivities.

Both robotic arms are fitted with a flexible cable management system, Teach Pendant armor, and screen protector obtained from MSITech. The flexible cable management should should typically be fitted with BNC cables to connect to. Visit MurrPlastik's website (https://www.murrplastik.com/products/robotic-dress-packs/fhs-flexible-holder-systems-for-collaborative-robots/) for videos on how to safely replace BNC cables and adjust this system. Whenever cabling is moved around, ensure that cables are managed neatly with consideration to avoiding any cross-talk especially to transducer BNC cables.

C.2.1 Robot Control

Ægir and Rán can be manually controlled via the attached teach pendant powered by the built in Polyscope software. This allows for precise motion in Cartesian coordinates relative to a defined TCP including orientation control using the robot's six joints of motion. The manual control is performed by simply pressing arrow keys on the teach pendant's manual motion control.

The robots may also be controlled via programmable motion performed directly on each robot's teach pendant. The programming is made for repeatable motion via way points, orientation, and various types of motion typically utilized in factory-like applications requiring high precision. Motion can be done as linear motion, j-motion or circular motion. Simplified grid pattern motion can be performed in preset routines such as a palletizing routine.

To perform this motion safely, the Polyscope installations must be programmed with the tool being used and safety parameters input in order for the robot software to fully understand how the robots should move. Inputs includes tool mass, tool center of mass, tool TCP position offset, joint limits, speed limits, acceleration limits, plane limits, and more.

Setting Joint limits can be the easiest method to prevent parts of the robot from crashing into anything. This is also beneficial to prevent Rán from colliding into the north tank wall, even when the Vention MachineMotion 7th axis extender moves to the end of the track.

The best method for creating safety parameters is to create safety planes, establish a tool position, and the parameters for the TCP. Safety planes should be made relative to the base or another feature. Create a new feature following the right hand rule (not according to the Teach Pendant or tank axes), where the thumb is Z+, the index finger is X+, and the middle finger is Y+, such that the Z+ direction is toward increasing danger or away from the safe operating positions. Copy the tool position from the established TCP. Safety for each plane should be named and set for "Normal" which will stop the robot. An additional Displacement of at least -50 mm should be set for each safety plane to ensure no damage to the hydrophones even with anechoic paneling in place. Safety parameters should not be set with a transducer attached in case of damage to the transducer. It is highly recommended, when moving the TCP near an object of interest to create a feature, that you move it manually both to and away from that feature. This will ensure a natural MoveJ of the robot does not make the tool accidentally crash into the new feature.

Ægir should have at least 5 safety planes and 2 joint limits when operating inside the tank. These include: Seabed or tank floor, X_{min} (Acrylic Wall), X_{max} (Acrylic Wall), Y_{max} (Acrylic Wall), and Lab Ceiling. An additional Y_{min} (Imaginary plane dividing robots) could be created to further prevent Ægir from hitting Rán. Ægir should have both the base and the shoulder joints limited to prevent the robot from colliding with the X_{min} (Acrylic Wall or East wall) as best as possible. The base joint limits can also benefit the prevention of Ægir hitting Rán.

Rán should have at least 4 safety planes and 2 joint limits when operating inside the tank. These include: Seabed or tank floor, $Xmin$ (Acrylic Wall), X_{max} (Acrylic Wall), and Lab Ceiling. Additional Y_{min} (Acrylic Wall) and Y_{max} (Acrylic Wall) may be set up in the installation. However, features are created relative to the base of the UR10e robot and because of the MachineMotion 7th axis extender, these planes can move relative to the track. In any case, care must be taken in the programming to ensure Rán does not crash into obstacles in the Y-direction or X-Z plane. Rán should have both the base and the shoulder joints limited to prevent the robot from colliding with the X_{min} (Acrylic Wall or East wall) and the Y_{min} (Acrylic wall or North wall) when the Vention MachineMotion 7th axis extender is moved to the end of the tank.

C.2.2 Vention MachineMotion

For further motion with Rán the Vention Machine Motion adding a 7th axis of motion may be connected to a computer via the ethernet port of the machine motion labeled [redacted]. An ethernet to USB adapter is provided to allow connectivity to the Microsoft Surface Pro 7 in the lab.

Using a chrome browser, input the IP address [redacted] to access the manual control of the 7th axis extender track for Rán. (Some information has been redacted for security purposes)

LEDs on the MachineMotion box should be lit when operating. 3 green-yellow LEDs should illuminate if the proper safety settings are loaded and the motor is ready to move. MachineMotion will not allow control until safety parameters or a program with safety parameters are loaded properly into the Polyscope software. Blue indicator lights on the manual control interface will light up when the track is sensed by the inductive sensors on the Home and End positions. If both indicator lights are lit, the system does not recognize a safe program loaded and will not allow manual control of the track. You may also notice that if the Vention control panel on google chrome on the tank mounted Surface Pro is still loaded from a previous session, it may not be showing a correct position for one of two reasons. First try refreshing the page, this may fix a connection problem with the Vention itself. After sitting for some time the google chrome page may have just fallen asleep and disconnected, restarting the page will allow for reconnection to occur. Secondly try to move the Vention to the Home position. At the Home position the electromagnetic sensor activates and the system re-calibrates the zero position and will now report the position correctly again. If Vention is still not functioning, a shut down and restart of the robot system may be necessary. When shutting down the robot system, shut down both Ægir and Rán and then hard shutoff Vention on the MachineMotion control box. Let them sit shut-off for a short period of time and turn them back on. Ensure that when Ægir and Rán are turned back on that the proper installation is loaded so all safety settings are loaded and when Vention recognizes a proper installation, it will function again.

The Vention MachineMotion track brings a level of difficulty and potential for damages when programming. This is because programs are written relative to the UR10e base and not relative to the MachineMotion track base. The track simply adds increased motion. This can cause a poorly programmed TCP waypoint position to crash into the tanks North wall when Rán is moved to the End of the track.

The Vention can be controlled via the teach pendant by use of additional Vention URCap software. When programming with the Vention URCap, it is important to ensure that the maximum speed is no more than $100mm/s$ and maximum acceleration as $50mm/s^2$. This will ensure a consistent stopping position at home and end of the track as defined by the inductive sensors on either end sensing the attached metal spring's proximity which should be 3 mm from the electromagnetic sensor to the end of the spring on either home or end positions. The approximate 3mm separation can be exceeded when allowed to move too fast and the springs are present to allow some give if the sensor is crashed into.

When programming, ensure the position of the Rán TCP in a program will not place the end connector/hydrophone assembly in a position to collide with the north wall of the tank or Ægir on the other end in order to prevent damages as the track moves to either end. This can be achieved for example by ensuring the TCP position is above the top of the tank when moving the track to the End position. This may also be accomplished by setting plane and joint limits for Rán. Once effective plane and joint limits are set in the Rán's installation, this becomes less of a worry.

Both the robotic arms and the Vention MachineMOtion 7th axis extender may be controlled remotely via custom software developed at BYU using LabVIEW. This software, Easy Spectrum Acoustics Underwater or ESAU, sends script commands directly to the robots and MachineMotion interface via TCP/IP through Ethernet connection. Specific Polyscope and Vention script commands can be found on universal-robots.com and https://vention.io/docs/guides/socket-api-61 respectively.

C.2.3 ESAU Motion Control

ESAU has the ability to send commands to each robot to perform motion control in the same system as taking measurements. This also allows for motion to only occur once the measurement has been taken and vice versa. ESAU reads in position and orientation statements from each robot through TCP/IP and passes commands to each robot arm informing them of position and orientation to move to. These commands are input as individual Cartesian tank coordinates or as multi-point scan positions. ESAU uses hard-coded coordinate translation based off of Rán's coordinate position relative to Ægir's relative to the tank. This minimizes error by not compounding multiple coordinate translations. The ability to measure this translation exactly greatly determines the true level of positioning precision that ESAU reports in tank coordinates.

Multi-point scan positioning allows for the selection of Cartesian coordinate limits for a scan, as well as number of positional points that should occur between those limits. However, hardcoded into ESAU, are maximum reachable positions creating a 3D polygon shape displayed when an input position is outside of that safely reachable range. Any multi-point scan position outside of the safety polygon may be deleted so the maximum amount of possible scan positions will be allowed for a scan measurement.

C.2.4 Robot End Connectors

The robot end connectors are used to attach the tool which holds the transducer. To maintain the precision of the robots, every possible measurement should be made, recorded, and clearly shown in a proposed design submitted to the machine shop. This way they can effectively make the design.

The end connectors should be attached to the robot arm using 4x Vented Socket Head Screw M6 x 1mm Thread, 25 mm long bolts that can be ordered from McMaster-Carr. Length of bolt may differ with end connector design.

After attaching the end connector and tool, measure the new TCP offset positions and program this into a new Polyscope installation with appropriate safety parameters and use the creation date in the name. Safety parameters should all be set in reference to the robot base coordinates and the tool end should extend 500mm in the y direction from the robot's end connection for robot safety parameters to take the TCP into account and to maintain the functionality of ESAU motion limit settings.

Each tool is equipped with a float sensor as an extra safety precaution to ensure neither robot touches the water. These sensors are wired into both robots as a safeguard stop. The sensor is oriented and wired to be naturally off or low voltage when out of the water (Note that common float switches are normally open because they typically are powered when not floating, telling a system to continue to fill a tank or that a tank is empty). When the sensor reaches water level the ring will be pushed to the top of the sensor and the signal will be switched to high voltage causing the whole system to immediately stop. The robot cannot move out of this stopped position without manually unswitching the float and manually moving the robot (this may require assistance). If the sensor was triggered while running ESAU, then ESAU will have lost connection with the robots as their E-Stop was triggered. If this is triggered by only Ægir, then Rán will have E-Stopped as well, but Vention may not have.

C.2.5 Robot Maintenance

Software updates should be regularly obtained at universal-robots.com/download. A new USER MANUAL version should be downloaded to Dr. Neilsen's BOX account under "Papers on Underwater Acoustics" with every software update. Download the software update onto a flash drive (Hydra is commonly used) and then load that onto each Teach Pendant.

When the Polyscope or ESAU software is updated, ensure that extensive testing is performed with any desired motion/programming in order to ensure no bugs have developed due to the update.

C.3 Other Laboratory Equipment

C.3.1 Depth and Temperature Sensors

The tank is equipped with two LMP 307T sensors from SensorOne, each mounted to the tank via suction cup mounts. These sensors improve remote measurements with ESAU by outputting current water depth and water temperature at two opposite corners of the tank. This allows for guided input of the current water depth preventing ESAU from allowing robots to move to positions of potential water damage and current water sound speed due to temperature.

Without proper knowledge of the current water depth, water damage to the robotic arms is possible. However, float sensors are also mounted to the tool connection in order to further prevent allowing the robots to become damaged. The depth and temperature sensors are mounted opposite corners of the tank and held in place with strong magnets adhered to the walls of the tank. The black caps protecting the pressure (depth) sensor diaphram should rest in contact with the tank floor for best results. A gap in the anechoic paneling should be provided for this.

The sensors are then attached to an NI card mounted to the underside of the Vention track. Sensor Wiring for the SensorsOne LMP307 connected to the NI cards is as follows.

Sensor 2199435 should be wired as follows:

Supply P+ (White) - Ground

Supply P- (Brown) - A17

Supply T+ (Grey) - Ground

Supply T- (Pink) - A16

Sensor 2199436 should be wired as follows:

Supply P+ (White) - Ground

Supply P- (Brown) - A14

Supply T+ (Grey) - Ground

Supply T- (Pink) - A15

If water does make contact with any electronics, Immediately Shut Off electronics and unplug to ensure no current is flowing. Allow sufficient time for each component to dry thoroughly before turning back on and ensure to make special note of what happened and what was being done at the time of the incidence. In the note, propose what might be done to prevent this occurrence in the future.

C.3.2 Transducers

In the U117 lab, there are 6 major types of underwater transducers used for measurements. Each transducer should be used with appropriate impedance matching transformer relative to frequency as well as appropriate power or pre amplifier.

AS-1 reciprocal hydrophone

Best used in the frequency range 1Hz-100kHz. Two AS-1 hydrophones are available. However, no specific calibrated sensitivity was given for either hydrophone and the provided in-line preamplifiers do not currently appear to work reliably.

Brüel & Kjær 8103 reciprocal hydrophone

Best used in the frequency range 4-100kHz. Seven 8103 hydrophones are available and are commonly used. These require use with the Brüel & Kjær NEXUS conditioning preamplifiers and specialized connectors/cables.

Teledyne Reson TC4038 reciprocal hydrophone

Best used in the frequency range 50-500kHz. Five TC4038 hydrophones are abilable. Due to their size, they are best for creating arrays without adding as much acoustic scattering potential.

Teledyne Reson TC4034 reciprocal hydrophone

Best used in the frequency range 5-200kHz. One TC4034 hydrophone is available. As the largest hydrophone, greater consideration must be given to ensure the safety of the hydrophone which may reach just outside of the robot safety limits.

Pool Speaker

The pool speaker does not have a very flat response as a source because it was specifically designed for providing a reference for synchronized swimming in the audible range.

Geospectrum Particle Motion Detector

The Geospectrum Particle Motion Detector (inventory number 7100) is an expensive piece of equipment ($5000+) and as such should be treated with great care. For any questions, refer to this manual and the User Manual contained in the pelican case with the detector.

Never suspend the particle motion detector from the cable. Always support with polyethylene rope from eyelet attachment on top of detector bought for this specific purpose. Use a sturdy knot that is tested not to slip. Fisherman's knot is proven to work, but another sturdy knot will suffice. The analog c/v converter should never get wet. There is a power supply with a fuse on the end that plugs into the wall that was specifically made for the analog c/v converter. The use of any other power supply is possible but not recommended. Refer to the particle motion detector user manual contained in the pelican case for specifications and limitations for the power supply. Similarly, the use of the particle motion detector without the included cable or analog c/v converter is possible but not recommended. If the need or desire ever arises, refer to the user manual contained in the pelican case for important instructions, and good luck. Never attach cable to detector or c/v converter without first applying Molykote 44 lubricant.

C.3.3 Data Acquisition

Data acquisition and signal generation is performed primarily using hardware from Spectrum Instrumentation and ESAU. The data acquisition cards have relatively high resolution (16-bit) and high sampling rate (40 MS/s). Using the Star-Hub module, the arbitrary waveform generator (AWG) (M2p.6546-x4) and digitizer (M2p.5932-x4) are accurately synchronized while housed inside an external PCIe chassis. As implemented, this configuration allots 128 mega samples for each of the four input and four output channels for each of two chassis that may be daisy chained for larger arrays. The chassis are connected to the control machine with ESAU software or other chassis via Thunderbolt 3 ports. Data is saved to the machine as binary files with associated text files detailing measurement settings for later processing via Python3 code developed in the research group.

Data acquisition may also be performed using AFR (Acoustic Field Recorder) also developed by BYU specifically for rocket/jet noise field tests. A Tektronix TBS 2000B Series digital oscilloscope is also available in the lab to perform more direct data analysis and acquisition.

C.3.4 Power Amplifiers

TEGAM 2350 Power Amplifier

The data sheet and specifications for the TEGAM 2350 high voltage power amplifier can be found on TEGAM's website (https://www.tegam.com/shop/signal-source-amplifier/2350/2350precision-amplifier-high-voltage-two-channel/). To avoid clipping or tripping the internal fuse, do not exceed ±4V input. This power amplifier provides a x50V output, such that the maximum ±4V input equals ±200V output. The maximum output current of this amplifier is 40mA, which is the real determining factor that depends on frequency due to each hydrophones frequency dependent impedance. Impedance matching transformers have been custom designed and built in house to attach between a source transducer and the TEGAM power amplifier to provide a more flat response depending on the desired frequency range. Ensure to take note that you are using the appropriate impedance matching transformer for the appropriate transducer in the appropriate bandwidth. This amplifier may be used with all the AS-1, Brüel & Kjær 8103, TC4038, and TC4034 transducers.

Teledyne RESON EC6081 mk2 VP2000 preamplifier

The Teledyne Reson EC6081 hydrophone preamplifier specifications can be found on Teledyne's website (http://www.teledynemarine.com/reson-ec6081) for basic details.

The preamplifier is connected to Teledyne receiving hydrophones to boost the signal before being sent to the DAQ. It is important to note that after the preamplifier is turned on, any settings are changed, or any connections are changed that you must let the preamplifier sit all ready to go for 30 seconds before taking measurements in order for the capacitors to charge properly. If not, measurements will not be trustworthy. Once this occurs the Teledyne does not need any extra wait time between scans until another change is made.

Brüel & Kjær NEXUS Conditioning Amplifier

The NEXUS conditioning amplifier is a preamplifier that can filter, add gain, and adjust for noise on a measured signal before that signal is sent to the DAQ. This specific conditioning amplifier is designed for charge sources with 2 channels in and 2 channels out. The input takes a special adapter that should be kept on the amplifier at all times. Two of the conditioning amplifiers were purchased specifically for the Brüel & Kjær 8103 hydrophones.

When using the Transducer SETUP screen, the default units will be pC/ms^{-2} which are picoCoulombs per meter per second squared. This is clearly an odd unit for our application but is great for use of an accelerometer, that can be changed by navigating over to Sensitivity, selecting down the column to the desired channel and pressing the +/- button to change the units to pC/Pa which are picoCoulombs per Pascal which is the appropriate charge sensitivity that is needed to be input for each 8103 hydrophone. The single-valued charge and voltage sensitivity of transducers should be assumed good for where the frequency response of the transducer is flat and can be found on the tan card in their respective cases. This change of units will not necessarily change the electrical output values setup with the default, unless the numeric value is changed, but it will lead to less confusion when using the conditioning amplifier.

The NEXUS manual indicates that charge transducer cables should be fitted with ferrite magnet cable clamps (provided) to reduce EMI intrusion only if the NEXUS amplifier itself is grounded and the channels are set to floating. In addition, the manual also advises to wind the cable twice around the ferrite core so as to form two loops, and the ferrite should be located as close to the NEXUS input socket as possible. This creates an inductor on the line to reduce high frequency noise. The ferrite magnets may also be used on the power supply line input if desired.

C.3.5 Signal Generation

Signal generation can be performed with software such as ESAU or AFR. It may also be performed by the Koolertron signal generator or the Brüel & Kjær precision 4063 80MHz dual channel function/arbitrary waveform generator.

C.3.6 Remote Access

Using a VPN and Remote Desktop, you may access the machines in the lab using the research team login information. Use of the Global Protect VPN will be necessary when accessing the lab computers using the shared research team account “panda-underwater” in order to gain off-site authorization. Do not select the use of remote desktop gateway when using the VPN. When logging in remotely with your personal byu login you may not be required to use the VPN instead connect via the remote desktop gateway similarly to how you would access the computers remotely as you normally would any machine on campus via remote desktop and your personal byu credentials.

When you plan to take measurements in the tank remotely, ensure you have either setup the amplifiers, preamplifiers, turned on the robots, etc. that you will not have access to control off site. You may use the OBS Studio software that is on the main Atlantis desktop in order to easily access all 4 webcams in the laboratory in order to visually monitor your experimentation live. This may also be easier if the lights are on in the lab.

C.4 Laboratory Processes

C.4.1 Taking Measurements with ESAU

Easy Spectrum Acoustics Underwater (ESAU) was created specifically for this laboratory as a sister program to ESTR (Easy Spectrum Time Reversal) used by Dr. Anderson's students and also created by Adam Kingsley. Its basic functionality runs in conjunction with ESTR. Before starting ESAU, ensure that the chassis are connected to the computer via Thunderbolt3 cable so the Spectrum cards may be initialized at startup.

The default card0 output setting is 300mV. This is the minimum required and bits will be lost if set below this even though it can be done. The Cards can output 12V, but should not be set this high. Stay between 100-6000 mV (ESAU will limit capabilities here). *It should be noted that 4V output from ESAU is the greatest the TEGAM power amplifiers will handle before clipping the signal or tripping a breaker. Often times, 3V should be sufficiently high.

Receiver cards should be setup with high impedance($1M\Omega$) to get voltage measurements and have negligible current. The low impedance setting works well when connecting to the TEGAM monitor output in order to monitor the signal you are generating and sending out from the TEGAM power amplifier.

The data that is recorded in .txt file contains the source and receiver positions recorded each as 12 numbers $(x, y, z, rx, ry, rz, x_{tcpoffset}, y_{tcpoffset}, z_{tcpoffset}, rx_{tcpoffset}, ry_{tcpoffset}, rz_{tcpoffset})$. For the "Scan Positions.txt" file, each row is a new scan position. A log.txt file is also generated for each scan position. The log.txt file contains the scan position, temperature from each of 4 sensors, card settings, transducers selected, and the signal configuration. Data recorded in .bin files are the raw recorded data saved as Float-32 single precision voltages(pressure when sensitivities are applied).

ESAU can generate infinite types of signals since a custom signal may be loaded into it. Typically simple sine waves and chirped signals (both linear and logarithmic) are used. Pulse signal generation is less understood within ESAU since it is not commonly used(this feature has not been explored much as of 11/4/2021). With pulsed signals, attenuation denotes the amount of time it takes for the signal to dissipate (in the frequency domain). This is the inverse of what we would otherwise expect it to be; increasing the attenuation decreases the width of the pulse in the time domain. For more information, visit https://zone.ni.com/reference/en-XX/help/371361R-01/lvanls/gmsp/.

Before taking first measurements in experiment, the following checklist will be useful:

- All cables running from transducers (hydrophones) are correctly connected and plugged in through filters and amplifiers to the DAQ and acquisition computer.
- Ensure that the cables are not crossing other cables which may cause induced noise. If they must cross, it is best to ensure that they cross at a 90° angle.
- Appropriate impedance matching transformer is attached or not (below 10kHz).
- Each component of the measurement chain is powered on. This may include checking to see if the TEGAM power amplifier has tripped a breaker.
- When using AFR (instead of ESAU), ensure it is reading data from correct ports associated with the transducers (if using multiple-card chassis pay close attention to card number).
- Ensure the sampling frequency is appropriate for equipment (below sampling maximum threshold) and experiment (high enough to accurately read frequencies of signal. This should

be set as a minimum with respect to the Nyquist frequency). For ESAU, you are not able to sample any lower than 128 kHz.

- Ensure the output voltage is not too high as to blow out transducers. The TEGAM amplifier should do a decent job at preventing this since it will trip a breaker and clip signals when the signal is 4V or more.

In the red ‘Signal’ section of ESAU, turn on the channel(s) being used for output into the tank and input your value of input voltage. To generate a signal click on the ‘Load/Gen Signal’ button. In the Load/Gen Signal you can input the desired Sampling Frequency (must be 128kHz or more). On the Generate page choose your Signal Type, frequency parameters, signal length including leading and trailing 0’s. Then click the ‘Generate’ button to load the signal. Once a signal has been loaded click Accept on the main page to use this signal.

In the blue ‘Recorder’ page turn on the channel(s) being used for recording signal data. Select the voltage sensitivity and a ‘High’ or ‘Low’ impedance setting (typically high for recordings and low for monitor signals). Only turn on the combined channel switches(large bar connecting two channels) if using Differential mode.

Select motion control for the UR10e robot arm control panel in ESAU. You must connect each robot and the Vention track. When connected, two green LEDs will be lit. You will also select which transducers are being use in this screen. A transducer must be selected in order to apply the small positioning offset to the acoustic center of the transducers. Manual motion allows you to move each individual robot to a chosen position in Cartesian tank coordinates. You may also choose a scan grid to apply by selected minimum and maximum Cartesian coordinates as well as number of points to iterate through. If the selected position(s) do not fit within the reachable limits that have been hard programmed into ESAU, you may select to dump the unfitting positions in order to maintain a large scan over the maximum reachable positions.

C.4.2 Laboratory Journals and Documentation

It is important to maintain a detailed record of experimentation at all times. This record used to be kept physically in a white with pink polka dots Lab Measurement Journal and Tank Maintenance Log. Records are now kept digitally and saved within each measurement folder along with saved ESAU .txt log files. There is a formatted .doc file on the Atlantis panda-underwater account desktop to copy and use for the record. A separate copy should also be found on Box.

Information to always record may include:

- Date and Time
- Researchers involved in measurements
- Goals for session of research
- Note Specific Test variations
- Note anything special noticed with each measurement
- Specific equipment used
- Transducers (including catalog ID)
- Amplifiers
- Filters
- Signal Generation
- Current water conditions
- Depth in meters

- Temperature in degrees Celsius
- Data file path
- Individual measurement ID's
- Improvements/Things to buy
- Summary of measurements taken and which may be valuable and what they may be valuable for

When recording information from measurements, it is important to remember that weeks or months later you will likely not remember the details of the measurement. The more detail you record, the easier it is to use the information or write about it later.

Measurement Chain

The automated measurement system can be used with any transducers assuming a robot end connector has been designed for the specific transducer and desired transducer orientation. Currently, we are using Aquarian Scientific AS-1, Brüel & Kjær 8103 phase matched, Teledyne Reson TC4034 [31] and Teledyne Reson TC4038 [30] hydrophones as both source and receiver transducers (Reciprocal transducers can act as both a projector and a receiver. An absolute calibration of reciprocal transducers may easily be obtained with 3 transducers.) for their relatively flat response up to 100 kHz, 4-100 kHz, 5-300 kHz and from 100-500 kHz respectively. Each transducer has a custom designed mount on a thin rod extended from the robot to maintain orientation of the transducers and protect the robot from water damage. The received signals are passed through a preamplifier (Teledyne Marine Reson VP2000 EC6081 mk2) [31].

The custom mounts allow for multiple transducer configurations including an added wire thermocouple to gather current localized environment conditions [11] without significant increased scattering. Currently, environmental conditions are acquired with two SensorsOne LMP 307T temperature and pressure/depth sensors from MCT RAM (mctram.com) sensitive from 0-86°F and up to 250 m depth in water pressure.

The cables connecting transducers to the DAQ run along the length of the robotic arms and may require special consideration of shielding to reduce induced noise from robot motors and brakes. Shielded RG58 coaxial cables were found to be sufficient for current applications. Higher levels of shielding or the use of shielding tape may also be used to further decrease noise. Noise levels were confirmed by mounting the preamplifier and conditioner to the receiving transducer before passing the signal along the robot arm.

The output signal from the arbitrary waveform generator (AWG) (ESAU signal generation) is passed through a power amplifier (TEGAM Model 2350). The TEGAM allows a maximum of 4 Vpp input and provides a gain of x50. The TEGAM output is passed through a transformer fabricated to address the impedance mismatch often found between an amplifier and a piezoelectric source [22]. This impedance matching transformer must be designed and built specifically for each transducer's impedance response.

The spectrum cards take in electronic signals from amplifiers via BNC adapters. The two spectrum units can be daisy chained via Thunderbolt3 cables, which is also how they communicate with the desktop.

As a preamble to a series of measurements, a calibration measurement is made to ensure proper signal analysis. During the calibration measurement, the source and receiver are positioned close together in order to reduce transmission loss through the water while also accounting for any minor phase effects. The response of a chirp covering the frequencies of interest is then broadcast and recorded. This calibration measurement can be used with known transducer response curves to ensure the frequency response of the measurement chain is taken into account. This through-the-sensor calibration incorporates the sensitivities of unknown components [7, 27]. The frequency response can be obtained by the time-gated response of the cross correlation or by a phase-corrected deconvolution [27, 35, 37].

Analysis code has been developed in Python for ease of use. See the git repository and Appendix B for details.

Bibliography

[1] U. S. G. S. . Mysid, “Diagram of a side-scan sonar,”, 2007.

[2] M. A. Ainslie, “A Century of Sonar: Planetary Oceanography, Underwater Noise Monitoring, and the Terminology of Underwater Sound,” Acoustics Today **11,** 12–19 (2015).

[3] S. L. Garrett, in *Understanding Acoustics*, M. F. U. o. T. a. A. Hamilton *et al.*, eds., (Springer, Pine Grove Mills, PA, USA, 2017).

[4] W. Kuperman and P. Roux, in *Springer Handbook of Acoustics*, T. Rossing, ed., (Springer New York, New York, NY, 2007), pp. 149–204.

[5] V. G. Jayakumari, R. K. Shamsudeen, R. Ramesh, and T. Mukundan, “Modeling and validation of polyurethane based passive underwater acoustic absorber,” The Journal of the Acoustical Society of America **130,** 724–730 (2011).

[6] P. Papadakis, M. Taroudakis, F. Sturm, P. Sanchez, and J. P. Sessarego, “Scaled laboratory experiments of shallow water acoustic propagation: Calibration phase,” Acta Acustica united with Acustica **94,** 676–684 (2008).

[7] J. D. Sagers and M. S. Ballard, “Testing and verification of a scale-model acoustic propagation system,” The Journal of the Acoustical Society of America **138,** 3576–3585 (2015).

[8] L. Zhang and H. L. Swinney, “Sound propagation in a continuously stratified laboratory ocean model,” The Journal of the Acoustical Society of America **141,** 3186–3189 (2017).

[9] K. L. Gemba, Doctor of philosophy dissertation, University of Hawai’I at Manoa, 2014.

[10] P. R. Molina, J. S. Rebull, C. O. Anglés, and N. Ortega, “Method for the Acoustic Characterization of Underwater Sources in Anechoic Tanks Based on Simulated Free-Field Scenario,” Instrumentation viewpoint pp. 70–73 (2015).

[11] L. T. Rauchenstein, A. Vishnu, X. Li, and Z. D. Deng, “Improving underwater localization accuracy with machine learning,” Review of Scientific Instruments 89 (2018).

[12] J. M. Collis, W. L. Siegmann, M. D. Collins, H. J. Simpson, and R. J. Soukup, “Comparison of simulations and data from a seismo-acoustic tank experiment,” The Journal of the Acoustical Society of America **122,** 1987–1993 (2007).

[13] J. Yangzhou, Z. Ma, and X. Huang, “A deep neural network approach to acoustic source localization in a shallow water tank experiment,” The Journal of the Acoustical Society of America **146,** 4802–4811 (2019).

[14] P. C. Etter, “Advanced applications for underwater acoustic modeling,” Advances in Acoustics and Vibration 2012 (2012).

[15] E. K. Westwood, C. T. Tindle, and N. R. Chapman, “A normal mode model for acousto-elastic ocean environments,” The Journal of the Acoustical Society of America **100,** 3631–3645 (1996).

[16] R. Kirby and W. Duan, “Modelling sound propagation in the ocean: A normal mode approach using finite elements,” Australian Acoustical Society Annual Conference, AAS 2018 pp. 530–539 (2019).

[17] D. F. V. Komen, Masters thesis, Brigham Young University, 2020.

[18] T. B. Neilsen, C. D. Escobar-Amado, M. C. Acree, W. S. Hodgkiss, D. F. Van Komen, D. P. Knobles, M. Badiey, and J. Castro-Correa, "Learning location and seabed type from a moving mid-frequency source," The Journal of the Acoustical Society of America **149,** 692–705 (2021).

[19] C. D. Escobar-Amado, T. B. Neilsen, J. A. Castro-Correa, D. F. Van Komen, M. Badiey, D. P. Knobles, and W. S. Hodgkiss, "Seabed classification from merchant ship-radiated noise using a physics-based ensemble of deep learning algorithms," The Journal of the Acoustical Society of America **150,** 1434–1447 (2021).

[20] D. F. Van Komen, T. B. Neilsen, D. B. Mortenson, M. C. Acree, D. P. Knobles, M. Badiey, and W. S. Hodgkiss, "Seabed type and source parameters predictions using ship spectrograms in convolutional neural networks," The Journal of the Acoustical Society of America **149,** 1198–1210 (2021).

[21] D. C. Baumann, J. M. Brendly, D. B. Lafleur, P. L. Kelley, R. L. Hildebrand, and E. I. Sarda, "Techniques for Scaled Underwater Reverberation Measurements," In *Oceans 2019 MTS/IEEE SEATTLE*, pp. 1–5 (IEEE, Seattle, WA, USA, 2019).

[22] N. L. Weinberg and W. G. Grantham, "Development of an Underwater Acoustics Laboratory Course Development of an Underwater Acoustics Laboratory Course *," The Journal of the Acoustical Society of America **49,** 697–705 (1971).

[23] S. Takahashi, T. Kikuchi, and A. Ogura, "Measurements of underwater sound absorption coefficient by the reverberation method,", 1986.

[24] O. Robin, A. Berry, O. Doutres, and N. Atalla, "Measurement of the absorption coefficient of sound absorbing materials under a synthesized diffuse acoustic field,", 2014.

[25] N. Cochard, J. L. Lacoume, P. Arzeliès, and Y. Gabillet, “Underwater Acoustic Noise Measurement in Test Tanks,” IEEE Journal of Oceanic Engineering **25,** 516–522 (2000).

[26] R. A. Hazelwood and S. P. Robinson, “Underwater acoustic power measurements in reverberant fields,” In *Oceans 2007 - Europe*, pp. 1–6 (IEEE, Aberdeen, 2007).

[27] J. L. Kennedy, T. M. Marston, K. Lee, J. L. Lopes, and R. Lim, “A rail system for circular synthetic aperture sonar imaging and acoustic target strength measurements: Design/operation/preliminary results,” Review of Scientific Instruments 85 (2014).

[28] G. D. Curtis, “Wide-frequency response of type J-9 underwater sound projector in a typical experimental tank,” Journal of the Acoustical Society of America **65,** 826–829 (1979).

[29] Q. Li, J. Xing, R. Tang, and Y. Zhang, “Finite-Element Method for Calculating the Sound Field in a Tank with Impedance Boundaries,” Mathematical Problems in Engineering 2020 (2020).

[30] J. D. Sagers, “Results from a scale model acoustic propagation experiment over a translationally invariant wedge,” Proceedings of Meetings on Acoustics 22 (2015).

[31] Z. Deng, M. Weiland, T. Carlson, and M. Brad Eppard, “Design and instrumentation of a measurement and calibration system for an acoustic telemetry system,” Sensors **10,** 3090–3099 (2010).

[32] A. Novak, P. Cisar, M. Bruneau, P. Lotton, and L. Simon, “Localization of sound-producing fish in a water-filled tank,” The Journal of the Acoustical Society of America **146,** 4842–4850 (2019).

[33] B. Borowski and D. Duchamp, “Measurement-based underwater acoustic physical layer simulation,” MTS/IEEE Seattle, OCEANS 2010 (2010).

[34] A. Novak, L. Simon, and P. Lotton, “Synchronized Swept-Sine : Theory , Application and Implementation,” Journal of the Audio Engineering Society **63,** 786–798 (2015).

[35] A. J. Berkhout, D. de Vries, and M. M. Boone, “A new method to acquire impulse responses in concert halls,” Journal of the Acoustical Society of America **68,** 179–183 (1980).

[36] M. Müller-Trapet, “On the practical application of the impulse response measurement method with swept-sine signals in building acoustics,” The Journal of the Acoustical Society of America **148,** 1864–1878 (2020).

[37] A. Farina, “Advancements in impulse response measurements by sine sweeps,” In *Audio Engineering Society - 122nd Audio Engineering Society Convention 2007*, (Audio Engineering Society, Vienna, Austria, 2007).

[38] B. Van Damme, K. Van Den Abeele, Y. Li, and O. B. Matar, “Time reversed acoustics techniques for elastic imaging in reverberant and nonreverberant media: An experimental study of the chaotic cavity transducer concept,” Journal of Applied Physics 109 (2011).

[39] B. E. Anderson, M. Clemens, and M. L. Willardson, “The effect of transducer directivity on time reversal focusing,” The Journal of the Acoustical Society of America **142,** EL95–EL101 (2017).

[40] A. Farina, “Simultaneous measurement of impulse response and distortion with swept-sine technique.,” In , 5159 (Audio Engineering Society, Paris, France, 2000).

[41] “Acoustics – Measurement of room acoustic parameters–Part 1: Performance spaces,” Standard, International Organization for Standardization, Geneva, CH (2009) .

[42] “Acoustics – Measurement of room acoustic parameters–Part 2: Reverberation time in ordinary rooms,” Standard, International Organization for Standardization, Geneva, CH (2008) .

[43] A. D. Pierce, *Acoustics*, 3rd ed. (Springer, 2019).

[44] G. C. Eastland and W. C. Buck, "Reverberation characterization inside an anechoic test chamber at the Weapon Sonar Test Facility at NUWC Division Keyport," **030003,** 030003 (2017).

[45] M. A. Ainslie and J. G. McColm, "A simplified formula for viscous and chemical absorption in sea water," The Journal of the Acoustical Society of America **103,** 1671–1672 (1998).

[46] R. E. Francois and G. R. Garrison, "Sound absorption based on ocean measurements: Part I: Pure water and magnesium sulfate contributions," Journal of the Acoustical Society of America **72,** 896–907 (1982).

[47] R. E. Francois and G. R. Garrison, "Sound absorption based on ocean measurements. Part II: Boric acid contribution and equation for total absorption," Journal of the Acoustical Society of America **72,** 1879–1890 (1982).

[48] V. P. Simmons, "Sound absorption in sea watera)," Journal of the Acoustical Society of America **62,** 558–564 (1977).

[49] M. R. Schroeder, "New Method of Measuring Reverberation Time," Journal of the Acoustical Society of America **37,** 409–412 (1965).

[50] "Acoustics – Measurement of sound absorption in a reverberation room," Standard, International Organization for Standardization, Geneva, CH (2003) .

[51] "Electroacoustics – Octave-band and fractional-octave-band filters – Part 1: Specifications," Standard, International Electrotechnical Commission, Geneva, CH (2014) .

[52] "Acoustics – Application of new measurement methods in building and room acoustics," Standard, International Organization for Standardization, Geneva, CH (2006) .

[53] A. Novak, M. Bruneau, and P. Lotton, “Small-Sized Rectangular Liquid-Filled Acoustical Tank Excitation : A Modal Approach Including Leakage Through the Walls,” Acta Acustica United with Acustica **104,** 586–596 (2018).

[54] H. Medwin, “Speed of sound in water: A simple equation for realistic parameters,” Journal of the Acoustical Society of America **58,** 1318–1319 (1975).

[55] W. D. Wilson, “The Journal of the Acoustical Society Of America Volume 31 Number 8 August 1959:Speed of Sound in Distilled Water as a Function of Temperature and Pressure,” Journal of the Acoustical Society of America **31,** 1067–1072 (1959).

[56] A. B. Coppens, “Simple equations for the speed of sound in neptunian waters,” Journal of the Acoustical Society of America **69,** 862–863 (1981).

[57] V. A. Del Grosso and C. W. Mader, “Speed of Sound in Pure Water,” The Journal of the Acoustical Society of America **52,** 1442–1446 (1972).

[58] V. A. Del Grosso, “New equation for the speed of sound in natural waters (with comparisons to other equations),” Journal of the Acoustical Society of America **56,** 1084–1091 (1974).

[59] K. V. Mackenzie, “Nine-term equation for sound speed in the oceans,” Journal of the Acoustical Society of America **70,** 807–812 (1981).

[60] M. Greenspan and C. E. Tschiegg, “Tables of the Speed of Sound in Water,” Journal of the Acoustical Society of America **31,** 75–76 (1959).

[61] C. C. Leroy, S. P. Robinson, and M. J. Goldsmith, “A new equation for the accurate calculation of sound speed in all oceans,” The Journal of the Acoustical Society of America **124,** 2774–2782 (2008).

Index