INDOOR POSITIONING USING SYNCHRONIZED ULTRASONIC OFDMA SIGNALS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Julian Bartolone

December 2021

COMMITTEE MEMBERSHIP

TITLE:	Indoor Ultrasonic Positioning Using Synchronized Ultrasonic OFDMA Signals
AUTHOR:	Julian Bartolone
DATE SUBMITTED:	December 2021
COMMITTEE CHAIR:	Vladimir Prodanov, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Tina Smilkstein, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	John Oliver, Ph.D. Professor of Computer Engineering

ABSTRACT

Indoor Positioning Using Synchronized Ultrasonic OFDMA Signals

Julian Bartolone

This paper proposes a method of short-range indoor localization using differential phase measurements of synchronized two-tone ultrasonic signals in an Orthogonal Frequency Multiple Access (OFDMA) scheme. This indoor positioning system (IPS) operates at an ultrasonic frequency of approximately 40kHz and synchronizes using an infrared signal. The OFDMA scheme allows for a receiver to process the signals from multiple transmitters continuously without the signals interfering with each other. The phases of the signals are measured using Goertzel Filters, allowing for low-complexity frequency content analysis. A MATLAB simulation using the proposed localization method is performed using four transmitter nodes in the corners of a 2.5m x 2.5m room and a receiver node within. The designs for the synchronizing transmitter node and the receiver node are then implemented in hardware and tested at 22cm and 28cm. The work described in this paper found that the proposed IPS functions correctly in simulation, and the hardware implementation of the receiver and transmitter provides accurate distance measurements with variance as low as 0.05cm. This variance is on the same order of magnitude as the wavelength of the ultrasonic signals used. The hardware used in the implementation of this design is low-power, low-cost, and easy to implement, but it carries with it design tradeoffs. The main difficulty introduced by the hardware is the generation of imperfectly orthogonal signals due to a time-discretization error imposed by the clock of the transmitter's general purpose microcontroller. This error is theoretically and experimentally analyzed yielding closely matching values.

Keywords: Ultrasound, Infrared, Positioning System, OFDMA, Goertzel Filter

ACKNOWLEDGMENTS

I'd like to extend my sincere thanks to my advisor, Dr. Vladimir Prodanov, for his friendliness, wisdom, and patience in guiding me and teaching me throughout my thesis and before that in the classroom. I'd also like to thank Dr. Tina Smilkstein and Dr. John Oliver for their time and their participation in my thesis committee. Thank you, as well, to the rest of the Cal Poly Electrical Engineering faculty. The wonderful professors, lecturers, and administration provide their students with a high-quality education and foster a supportive environment that encourages curiosity, diligence, and integrity.

Finally, to my parents, my sister, my grandmother, my family, and my friends: I can't thank you enough for giving me the love and support needed to pursue my degree and life goals. I truly would not be where I am today without you all. Thank you Anna, for being the best lab partner I've ever had. Thank you Tristan, Kurt, Austin, Jason, and Nolan for being the best friends and roommates I could ever ask for. Thanks to all of you, my time at Cal Poly was fulfilling, enriching, and fun.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

1 Introduction

Positioning systems are highly integrated into modern life [1] [2]. The purpose of a positioning system is to let a user know the location of an object within a defined space without any physical attachment to the object. The most common example is the Global Positioning System (GPS) which uses satellite trilateration to measure an object's positioning on the surface of the Earth [3]. GPS is excellent for use in navigation systems in cars, planes, ships, smartphones, and many other outdoor, long-distance applications. However, it falls short when considering indoor, short-range scenarios like in a building [4]. This calls forth the need for an Indoor Positioning System (IPS) to allow for accurate object-tracking and navigation indoors. Imagine how helpful it would be if a system could give its users precise directions to a gate in an airport, to a vendor in a large convention center, or to a patient or doctor in a medical facility.

1.1 Motivation and Use Cases

The motivation for this IPS project is to provide a low-cost, low-complexity solution to indoor localization in applications where GPS can't provide accurate results. A system like this is useful in warehouses for tracking automated robots, in hospitals for tracking patients and doctors, in airports and large buildings of high traffic for indoor navigation, in consumers' homes for Internet of Things (IoT) purposes, and many more applications left up to the reader's imagination [5].

Some ultrasonic-based IPSs are currently available to purchase, with most targeting businesses and hospitals [6]. These include Cricket and Sonitor which use proprietary combinations of electromagnetic and ultrasonic signals. The former is a 2004 project from MIT which uses signal chirps between fixed ultrasonic and electromagnetic receivers and a moving transmitter. The Cricket system leverages the propagation delay between the ultrasonic and electromagnetic signals to calculate distance, as ultrasound travels much slower. It uses the same trilateration concept as GPS to measure the position of the transmitter using the distances to four receivers [7]. Sonitor, on the other hand, uses a building's existing Wi-Fi infrastructure in conjunction with custom 40kHz ultrasonic nodes to perform the task of indoor positioning [8]. However, both of these systems are complicated and more expensive than the design proposed in this thesis.

1.2 Project Scope

The hardware implementation explored in this work models the link between the synchronization transmitter node and the receiver node. The results of this link design are analyzed and extrapolated to model a hypothetical full system with four transmitter nodes and a receiver node. The system can easily scaled to include many receiver nodes to be localized. In future work, the same transmitter design can be repeated three times to create the full system, where each transmitter produces orthogonal two-tone ultrasonic signals ideally spaced at 100Hz, with design tradeoffs discussed in Section 4.

2 Background

This background section provides an introduction to the concepts used in the design. It starts with a more in-depth description of positioning systems and then transitions into describing the specific techniques used for this thesis, specifically the method for obtaining a distance measurement from a continuous ultrasonic signal. The chosen techniques are compared with other common methods used in similar systems. The goal of this section is to explain the math and derivations of the system.

2.1 The Global Positioning System

To begin the background information for this thesis, GPS is first explained to give a comparison against Indoor Positioning Systems. GPS functions via a network of 24 Medium Earth Orbit (MEO) satellites that continuously transmit their current time measured by highly accurate atomic clocks. Each measure of time is encoded into sinusoidal electromagnetic radio-frequency (RF) waves using a Code Division Multiple Access (CDMA) scheme, explained more in Section 2.4. The GPS receiver on the surface of the Earth collects these signals from at least four satellites and measures the difference between its current time and the time encoded in the received signals. Using this time difference, and since the satellites are in known locations along their orbits, the distance between the receiver and each satellite is calculated using the speed of the RF signal (approximately the speed of light, $2.99 * 10^8 \frac{m}{s}$) as in Eq. (2.1) [3].

$$d_{RF} = v_{RF} * (t_{RX} - t_{TX}) \tag{2.1}$$

Using the distances to the satellites, the receiver is able to compute its position by trilateration. Letting the Earth's surface be an ideal 2D plane, a receiver would need to process the signals from 3 satellites in order to obtain the unique triplet of distances that correspond to its current position on Earth. Using 4 satellites, multiple location measurements can be determined by trilateration and averaged to eliminate error associated with the equipment [1].

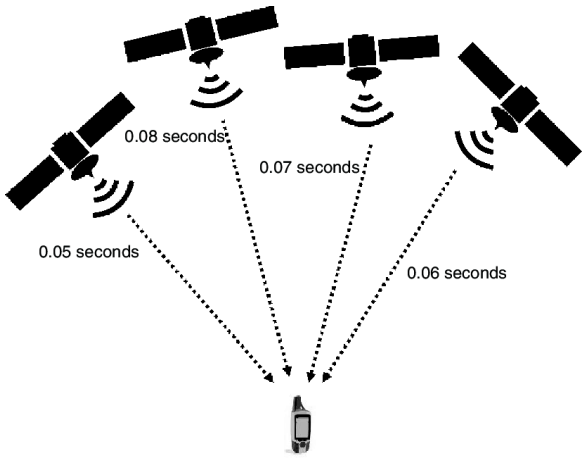

Figure 2.1: The Global Positioning System Receiver-Transmitter Distance Measurement [3]

Trilateration works by creating a circle centered at each transmitter with radius equal the transmitter's measured distance. The receiver's position is then the intersection of these three circles. This concept is illustrated in Fig. 2.1.

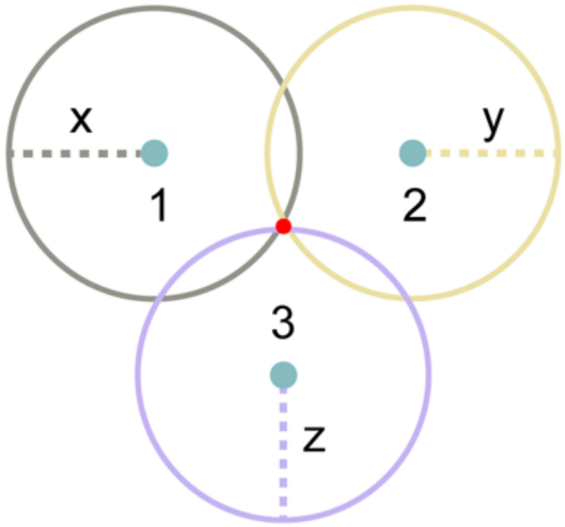

Figure 2.2: Trilateration Process [2]

With four transmitters, three trilateration measurements can be produced and averaged for a more accurate result in the presence of inaccuracies due to random system noise and jitter as well as receiver clock frequency. These concepts will be discussed in further detail in the following sections. In a typical GPS location measurement, more than four satellites are visible from the receiver, and an even more accurate result can be produced by averaging.

2.2 Indoor Positioning Systems

An Indoor Positioning System is a type of positioning system that, much like GPS, uses forms of traveling-wave-based communication to sense and approximate the position of an object, but in smaller space such as within a room. GPS fails indoors due to satellite signal attenuation by the construction materials of a building [9]. Also, GPS has an approximate accuracy of 5 meters which could span an entire room, rendering an indoor position measurement useless [3].

Some IPSs use digital wireless technology such as Wi-Fi or Bluetooth to measure Time of Arrival (ToA) or received strength of a signal and calculate an object's approximate location [10]. However, due to their high frequency and fast propagation speed of roughly the speed of light ($2.99 * 10^8$ m/s), these RF signals can penetrate walls thus causing interference between two adjacent rooms with IPSs installed. They also require more complex designs and more expensive materials [4] [11]. An alternative wireless communication technique that solves these problems uses ultrasonic signals.

2.3 Ultrasonic Signals

Ultrasonic signals are acoustic waves with frequency higher than the audible frequency spectrum for humans. Humans can typically hear from about 20Hz to 20kHz, so an ultrasound wave is defined as a sound wave above this frequency range [12]. Since it is a physical sound wave, an ultrasonic signal does not pass clearly through walls because its kinetic energy gets absorbed by any obstructing object. This means that an IPS based on ultrasonic technology will not have inaccuracies due to interference from adjacent rooms [10]. To determine the characteristics of an IPS based on ultrasonic technology, the velocities and wavelengths of the acoustic signals are highly important. This thesis assumes an ultrasonic velocity of 343 $\frac{\mathrm{m}}{\mathrm{s}}$, but in reality this value varies slightly due to the characteristics of medium of travel (in this case, air) [12]. The dominant characteristic of air that can affect the speed of sound is temperature, with the relationship shown in Eq. (2.2) where T is temperature in Celsius. [13]

$$\mathrm{v_{US}} = 20.05 * \sqrt{\mathrm{T} + 273.15}\ \mathrm{m/s} \tag{2.2}$$

And the wavelength of an ultrasonic (US) signal is given by:

$$\lambda = \frac{v_{US}}{f_{US}} \tag{2.3}$$

The ultrasonic transducers used in this project are inexpensive, widely available, and simple to implement [14]. They are designed to operate at 40kHz, so the wavelength in air becomes:

$$\lambda = \frac{343 \ \frac{m}{s}}{40 \text{ kHz}} = 8.6 \text{ mm} \tag{2.4}$$

2.4 Multiple Access Schemes

An important feature of this proposed IPS is that the transmitters can emit their signals continuously and simultaneously without interfering with one another. This concept, called signal orthogonality, is a requirement for a positioning system with multiple transmitters. There are three main methods of attaining this signal orthogonality.

As mentioned, GPS uses CDMA, a scheme in which each satellite has a unique code that it transmits along with it's atomic clock's time. The receiver can then use these codes to discern which received signal came from which satellite in order to perform position trilateration [3] [15]. CDMA effectively creates signals that are orthogonal in power. This the approach used in [16].

The second method is Time Division Multiple Access (TDMA) in which each transmitter of a system has an allotted time to send its signal, thus creating signals that are orthogonal in time. The receiver knows which signal came from which transmitter by measuring according to the transmitters' schedule. This approach is used in Cricket [7] and in [17].

Lastly, received signals can also be distinguished in the frequency domain by leveraging a Frequency Division Multiple Access scheme such as in [18], [10], and [4]. Fig. 2.3 illustrates these three methods. These related works are discussed in more detail in Section 2.10.

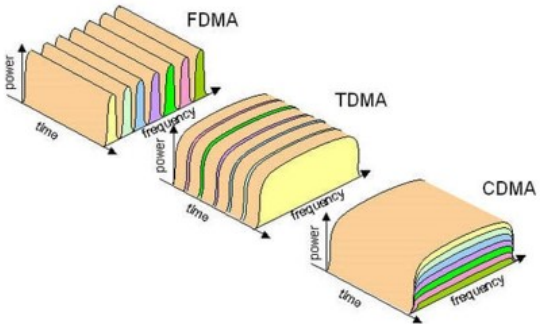

Figure 2.3: Multiple Access Schemes [15]

2.4.1 Orthogonal Frequency Division Multiple Access

Orthogonal Frequency Division Multiple Access (OFDMA) is a variation of FDMA used in wireless communication to send parallel data streams over multiple sub-carriers (tones) that are orthogonal in frequency and uniquely measurable [19]. Frequency orthogonality means that performing a Fast Fourier Transform (FFT) on the received signal allows for distinction between the sub-carriers because their spectra don't overlap, and thus each channel can be read without interference from the other channels [4] [16]. Fig. 2.4 shows the spectra of signals in an OFDMA system, where the peaks of one sub-carrier align with the nulls of all other sub-carriers.

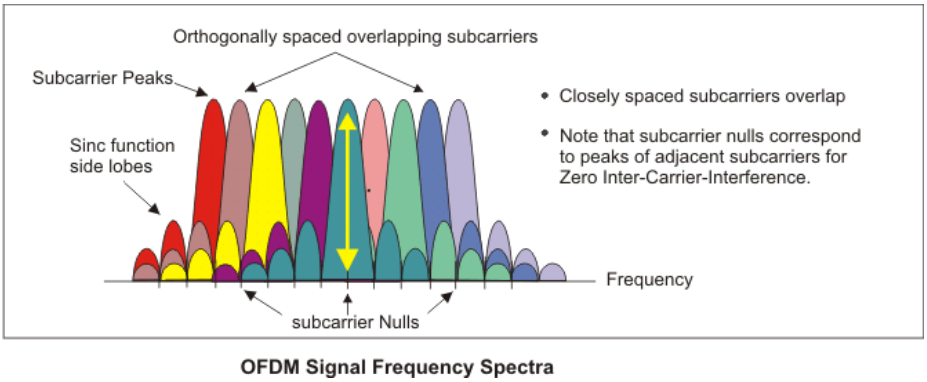

Figure 2.4: Orthogonal Frequency Division Multiplexing [20]

This thesis uses an OFDMA setup because it lends well to the method of distance measurement explained in the following section. It employs frequency orthogonality in two aspects: between the tones, or sub-carriers, of the two-tone ultrasonic signals of each transmitter and between the two-tone ultrasonic signals of the separate transmitters themselves. This allows for the distinction between the signals from each transmitter as well as the distinction between the two tones each of the signals at the receiver. One condition of this orthogonality is that the sub-carriers tones must be spaced evenly in frequency and evaluated in blocks that capture integer

numbers of periods of the tones. This concept is explained further in Section 2.6.

2.5 Distance Measurement

There are a number of techniques to determine an object's position from the received ultrasonic OFDMA signals [9] [10] [18]. One method is to measure the intensity of the received signals and make an estimatation of distance based on its magnitude relative to the generated signal. This estimation is based on the ideal wave magnitude's inverse square proportionality to the distance it travels [4]. However, there are many factors that affect the attenuation of the ultrasonic signal during generation, transmission, and reception. These factors can be nonlinear and difficult to model, and thus distance measuring using received signal intensity is not accurate enough for practical use. A more favorable method is to send two-tone, i.e. two-sub-carrier, orthogonal ultrasonic signals through each transmitter and calculate distance with the received relative phase shifts of the tones of the signals.

2.6 Calculating Distance by Phase Measurement

2.6.1 Single Tone Signaling

First, consider a system that uses a single-tone (sinusoidal) ultrasonic signal, a_1. The equation for this generated signal is shown in Eq. (2.6), where A_1 is the signal's amplitude, f_1 is the time frequency of the wave in Hertz, and t is the time elapsed since the signal was first generated with zero phase.

$$a_1(t) = A_1 \sin(2\pi f_1 t) \tag{2.5}$$

As the signal travels through its medium, air, it incurs a phase shift relative to the distance it has traveled. Thus, the equation for the received signal is as follows, where d is the distance traveled in meters, λ_1 is the wavelength of the signal in meters, and φ_{US} is a deterministic phase shift exerted on the signal by the filtering characteristics of the ultrasonic transducers (discussed in Section 4.1.1).

$$a_1(d,t) = A_1 \sin(\frac{2\pi d}{\lambda_1} + 2\pi f_1 t + \varphi_{US}) \tag{2.6}$$

By synchronizing the transmitter and receiver nodes in time using an electromagnetic signal traveling at the speed of light such as infrared, a concept explained further in Section 2.7, the equation becomes:

$$a_1(d) = A_1 \sin(\frac{2\pi d}{\lambda_1} + \varphi_{US}) \tag{2.7}$$

This signal's phase, Φ_1, is solely a function of distance:

$$\Phi_1(d) = \frac{2\pi d}{\lambda_1} + \varphi_{US} \tag{2.8}$$

Φ_1 can be measured via an FFT or alternative filter (explored in Section 2.8), thus producing a distance measurement:

$$d_1 = \frac{(\Phi_1 - \varphi_{US})\lambda_1}{2\pi} \tag{2.9}$$

However, the phase of a single tone will only be a value in the range [0, 2π]. For f_1 = 40 kHz, the maximum distance that can be measured with one tone without abmbiguity is (letting $\varphi_{US} = 0$ because it does not affect the maximum distance):

$$d_{1,max} = \frac{2\pi\lambda_1}{2\pi} = \lambda_1 = 8.6 \text{ mm} \tag{2.10}$$

So, at distances beyond 8.6 mm, the phase of the 40kHz ultrasonic signal and, therefore, the perceived distance resets to 0, thus creating ambiguity in the distance measurement. This range cannot support an IPS as it is not nearly long enough to cover the dimensions of a room.

2.6.2 Two-Tone Signaling

Instead, consider a system using a two-tone ultrasonic signal, a_{12} consisting of sinusoids a_1 and a_2 where upon generation (letting φ_{US} = 0 for simplicity's sake):

$$a_1(t) = A_1 \sin(2\pi f_1 t) \tag{2.11}$$

$$a_2(t) = A_2 \sin(2\pi f_2 t) \tag{2.12}$$

$$a_{12}(t) = a_1(t) + a_2(t) \tag{2.13}$$

Assuming the tones have equal amplitudes $A_1 = A_2 = A$, using the sum of sines trigonometric identity, the two-tone signal becomes:

$$a_{12}(t) = 2A \sin(2\pi \frac{f_1 + f_2}{2} t) \cos(2\pi \frac{f_1 - f_2}{2} t) \tag{2.14}$$

This appears as a sideband-suppressed amplitude-modulated (AM) signal with carrier frequency $f_c = \frac{f_1 + f_2}{2}$ and envelope, or beat, frequency $f_{12} = |f_1 - f_2|$ [21]. The two tones are orthogonal on this beat period, meaning they align to zero phase every beat period $T_{12} = \frac{1}{f_{12}}$.

As shown in Section 4.1.1, the tone frequencies for transmitter TX_a are 40kHz and 40.1kHz. Fig. 2.5 shows these two tones and their phases as well as their two-tone signal and its phase. These plots are only a function of distance, meaning time t is held constant and the waveforms are synchronized.

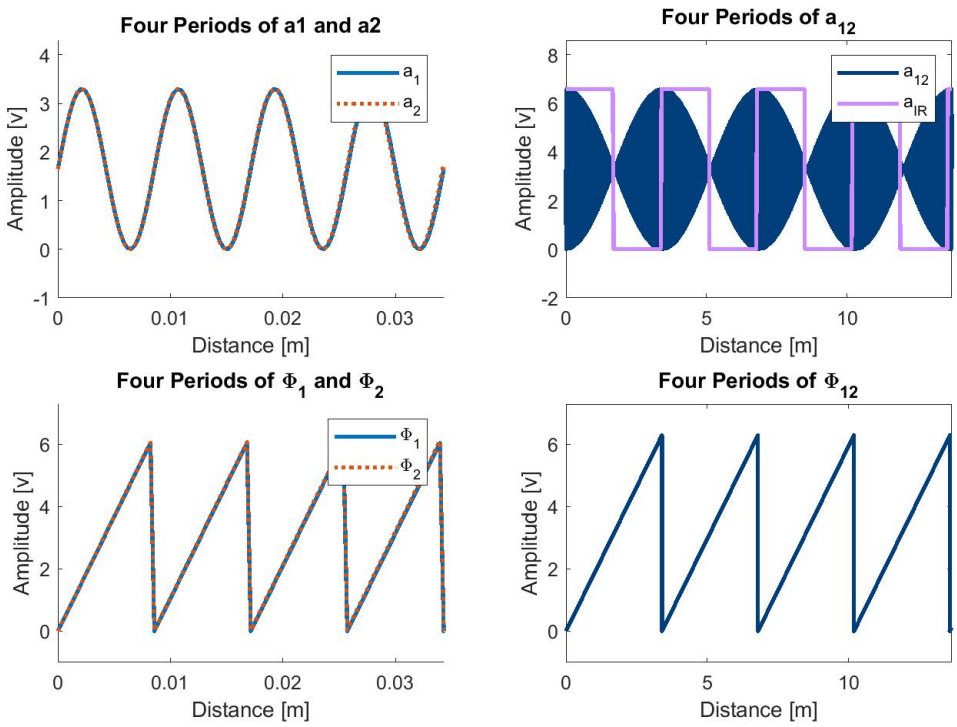

Figure 2.5: Calculating Distance by Phase Shift; f_1 = 40kHz, f_2 = 40.1kHz, f_{12} = 100Hz

The beat frequency, and thus the IR synchronization frequency, in this example is 100Hz. These two terms are used interchangeably throughout this thesis. With this frequency, the maximum measurable distance for each transmitter is calculated:

$$d_{12,max} = \frac{2\pi\lambda_{12}}{2\pi} = \lambda_{12} = \frac{343\ \frac{m}{s}}{100\ Hz} = 3.43\ m \tag{2.15}$$

This is shown in the plot of the Φ_{12}, where the phase measurements start to repeat after 3.43 m.

Therefore, by mapping every value of Φ_{12} from 0 to 2π to a unique distance from 0 meters to λ_{12} = 3.43 meters, a distance measurement can be obtained from the phase shifts of the two-tone signal. This mapping by a linear transform.

2.7 Receiver-Transmitter Time Synchronization

As mentioned previously, the receiver and transmitters must be synchronized in time in order to obtain a simple relationship between the phase measurements of the two-tone signal and distance or the receiver. This is because the two tones will experience phase shifting due to time delays, $\varphi_{\tau,1} = 2\pi f_1 \tau$ and $\varphi_{\tau,2} = 2\pi f_2 \tau$ calculated according to Eq. (2.16):

$$\varphi_\tau = 2\pi f \tau \tag{2.16}$$

Where τ is the time elapsed since the signals were first generated with 0 phase. These phase shifts are equal to 0 when τ is an integer multiple of the beat period of the two-tone signal. This means that the receiver can measure the phase of a signal with period equal to the beat period in order to measure the delay, τ. This signal is the synchronization signal, and it must not incur any phase shift due to distance traveled. This is because it must provide the receiver a measure of the phase shift of the two-tone signal *before* it travels in the air.

There are a number of techniques to attain this synchronization. One method is to plant anchor transmitters at known locations within an IPS's area. This eliminates the dependence of the phase shift on distance between these anchors, thus allowing the receiver to calculate the received time delay [18] [9]. Another method, the method used in this thesis, is to transmit the two-tone signal's phase via an Infrared link between the receiver and transmitter [22]. Assuming the travel time of the IR signal is negligible because it travels at nearly the speed of light, the IR signal will not be phase shifted due to distance, and will thus produce a measure of τ.

$$\tau = \frac{\Phi_{IR}}{2\pi f_{IR}} \tag{2.17}$$

Fig. 2.6 illustrates this concept. The top plots show the signals as received, meaning they are subject to phase shifts from elapsed time and distance traveled. In this example, the elapsed time is set to 3ms and the distance is set to 1m. The bottom plots show the result of subtracting the phase of the IR synchronization signal (shown in red) from all of the signals, thus eliminating time shift τ from the system.

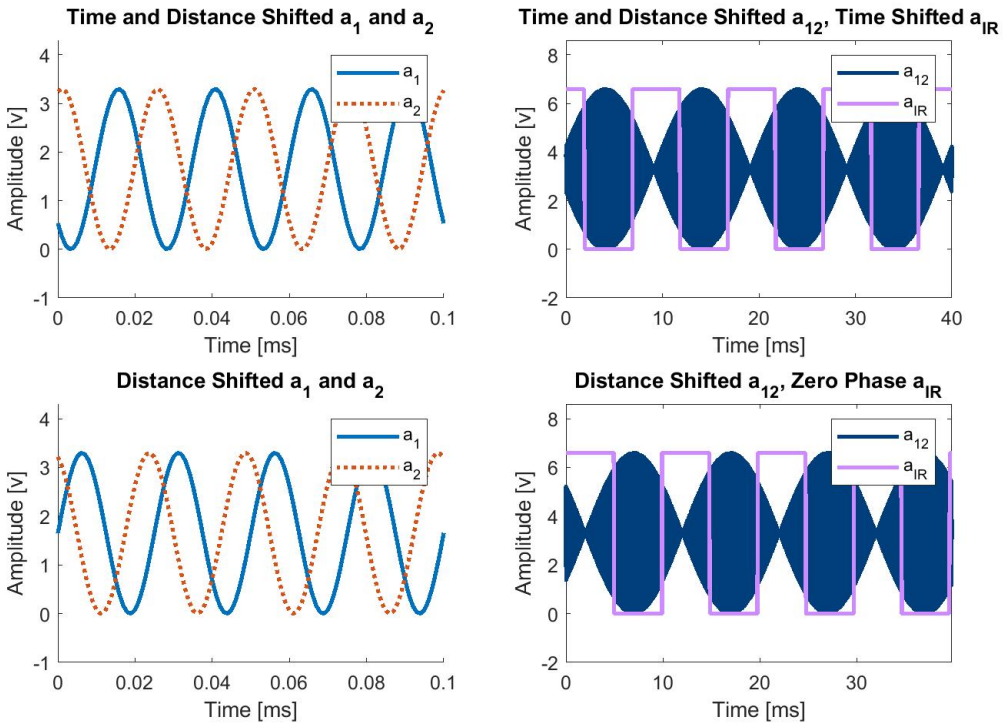

Figure 2.6: IR Synchronization Example

The phase shifts of the tones in the bottom plots of the figure are thus a function of only distance, and they can be extracted and transformed to yield a distance measurement.

2.8 Goertzel Filter

So far, the method for obtaining a distance measurement given the received phases of the tones of a two-tone ultrasonic signal and the received phase of an IR signal transmitting at the beat frequency of the ultrasonic signal has been established. Now, a method for obtaining these phases from the received signals is required.

The most popular method of analyzing a periodic signal for its frequency content is by using an Fast Fourier Transform implemented on a digital device [23]. The signal is sampled into a block of N samples using an Analog-to-Digital Converter (ADC) with sampling rate f_s. This block is then fed into the FFT to obtain the magnitude and phase of the signal at $\frac{N}{2}$ frequencies within a defined range.

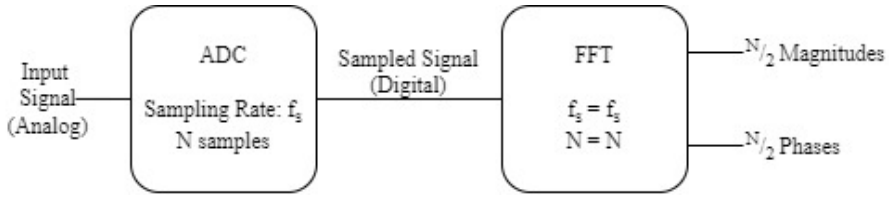

Figure 2.7: Fast Fourier Transform

However, the design proposed in this thesis only requires the analysis of two frequencies per transmitter plus one synchronization frequency for a total of nine frequencies for a 4-transmitter IPS. Therefore, using an FFT to compute the phases of the signals is wasteful in terms of power and speed because many unimportant frequencies are computed. This is especially relevant in a low-power, embedded environment such as the proposed design.

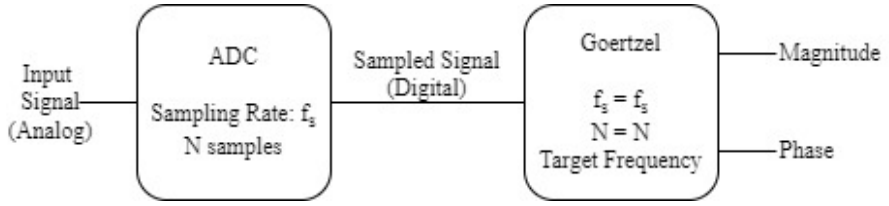

Figure 2.8: Goertzel Filter

To circumvent the issues associated with using an FFT, the design attempted in this work instead uses a Goertzel Filter [24], whose block diagram and transfer function can be seen in Fig. 2.9 and Eq. (2.18). The filter takes the form of a second-order inifinte impulse response (IIR) filter.

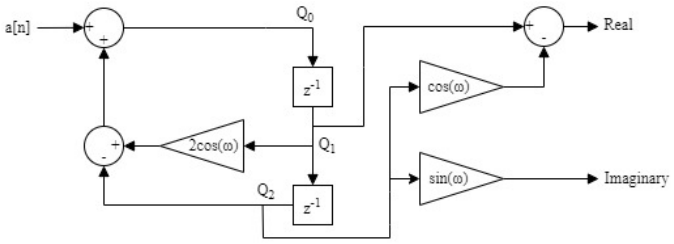

Figure 2.9: Block Diagram of Goertzel Filter

$$H(z) = \frac{1 - W_N^k z^{-1}}{1 - 2\cos(2\pi k/N)z^{-1} + z^{-2}} \tag{2.18}$$

The Goertzel Filter is quite simple in practice. Like the FFT, is accepts a block of N samples obtained from an ADC and iterates through each sample to compute the signal's frequency content. Unlike an FFT, how-

ever, it requires that the user specify the frequency to be analyzed. Whereas an FFT wastefully computes $\frac{N}{2}$ frequencies within its defined window [23], the Goertzel Filter only analyzes one target frequency. This essentially makes it a bandpass filter, allowing only the specified frequency to pass through for magnitude and phase analysis so long as the signals are properly orthogonal. With decreasing orthogonality, the Goertzel Filter resembles a bandpass due to spectral leakage from other tone frequencies. This is explored further in Section 3.

In order to maintain orthogonality between the two tones, as mentioned in Section 2.4.1, the block size, N, must capture an integer multiple of the period of the target signal [24]. Eq. (2.19) shows the formula used to calculate N that satisfies this requirement, where α is an integer representing the number of periods captured.

$$N = \text{int}(\alpha * \frac{f_s}{f_{target}}),\ \alpha = 1, 2, 3, ... \tag{2.19}$$

For example, to capture 3 full beat periods of a two-tone signal spaced at 100Hz with a sampling rate of f_s = 1MHz, N must be:

$$N = \text{int}(3 * \frac{1\text{MHz}}{100\text{Hz}}) = 30000 \text{ samples} \tag{2.20}$$

For comparison, an FFT performed on this sample set would yield $\frac{N}{2}$ = 15000 frequency measures, 14991 more than needed. Implementing nine separate Goertzel Filters tuned to the required frequencies eliminates this waste.

It is N that specifies the target frequency for the Goertzel Filter. After N is determined, two more constants are calculated:

$$k = \text{int}(0.5 + \frac{N * \text{Target Frequency}}{f_s}) \tag{2.21}$$

$$\omega = \frac{2\pi k}{N} \tag{2.22}$$

Then, the filter iterates through each sample in the block of N samples to perform the following computations. Here, a[n] represents the value of the current sample ($n : 1 \rightarrow N$), and the running parameters Q_0, Q_1, and

Q_2 are used to store values as the block is parsed. These running parameters have an initial value of 0.

$$Q_0 = 2\cos(\omega) * Q_1 - Q_2 + a[n] \tag{2.23}$$

$$Q_2 = Q_1 \tag{2.24}$$

$$Q_1 = Q_0 \tag{2.25}$$

When the whole block has been parsed (n = N), the frequency components are determined with the following formulas.

$$\text{Real} = Q_1 - Q_2 * \cos(\omega) \tag{2.26}$$

$$\text{Imaginary} = Q_2 * \sin(\omega) \tag{2.27}$$

$$\text{Magnitude} = \sqrt{\text{Real}^2 + \text{Imag}^2} \tag{2.28}$$

$$\text{Phase} = \tan^{-1}(\frac{\text{Imag}}{\text{Rreal}}) \tag{2.29}$$

After obtaining the phases of the two tones in the synchronized two-tone ultrasonic signal, the distance between the transmitter and receiver is determined via the aforementioned linear transform in Section 2.6.2.

2.9 Multilateration

Thus far, the process for obtaining the distance measurement between one transmitter and the receiver has been established. Applying the same procedure to the remaining three transmitters yields their distances to

the receiver as well, where the tone frequencies are shown in Table 4.1. Using these four distances, the position of the object is calculated via multilateration. This is similar to the trilateration process used by GPS described in Section 2.1. The exact method is discussed further in Section 3.1.

2.10 Related Works

This section briefly describes some related works in order to form a basis for comparison of the proposed system's performance.

The approach used in [4] is similar to this design, as it uses the differential phase shift of ultrasonic OFDMA signals to measure the distance between a transmitter and receiver. Its method of obtaining this distance measurement is by creating a fourth-order bandpass model of the ultrasonic channel formed by the same transducers and extracting its phase characteristics. Then, assuming the system is already time-synchronized, the channel model is used to map the received differential phases of a multitone signal into unique distance measurements. This related work provides a simulation of its design achieving a standard deviation in its distance measurements of about 5mm, which is on the same order of magnitude of the accuracy achieved in this work's simulations. However, a hardware implementation was outside of the scope of [4], so therefore it cannot provide a comparison for the hardware built in this thesis.

The IPS designed in [16] is also comparable, but it uses audible sound frequency ranges and a CDMA scheme, rather than ultrasonic frequencies and OFDMA. It was able to both simulate the IPS design and implement it in hardware. Much like the work done in this thesis, [16] attempts a fully embedded receiver but forfeits it for a more robust laptop computer during testing. With this, it was able to achieve an accuracy of 1mm to 10cm, depending on the receiver's position within the room. This is directly comparable to the lowest variance of 0.05cm found in the hardware implementation of this thesis. The drawbacks of this related work, however, are that by using audible sound frequencies, humans can hear the positioning signals and the transducers are much larger than those used in this thesis.

3 Simulation

Before the hardware of the IPS is explored, the system is first modeled in software using MATLAB. This simulation section explores both ideal and and real-world scenarios, which will be explained in further detail in a later section. The simulations performed yielded errors in the position measurement ranging from approximately 0.1mm to 1mm in the ideal case and approximately 1cm to 5cm. The MATLAB code used to produce these simulations can be found in the Appendix.

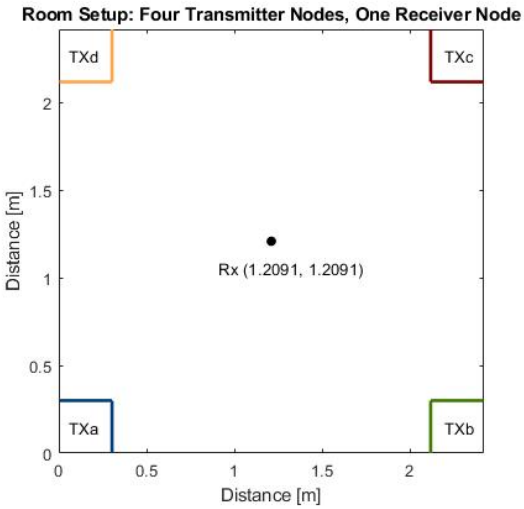

Figure 3.1: Proposed System in an Indoor Space

The setup in Fig. 3.1 depicts the full system, with four transmitter nodes placed in the corners of a room and an object with a receiver node placed in the middle of the room. Each transmitter node consists of an ultrasonic transmitter, with one of the nodes (TX_a) containing an IR transmitter as well. The receiver node contains one ultrasonic receiver and one IR receiver. Due to the OFDM scheme, the receiver can accept the two-tone ultrasonic signals from all four transmitters while still maintaining the ability to distinguish between them.

The dimensions of the room are chosen to be W = 2.42 m by L = 2.42 m. This length and width mean the diagonal of the room is:

$$\text{Diagonal} = \sqrt{\mathrm{W}^2 + \mathrm{L}^2} = \sqrt{(2)(2.42^2)} \simeq 3.42\text{m} \tag{3.1}$$

This ensures that each transmitter can provide accurate distance measurements throughout the entire room, given the maximum distance of one transmitter (3.43m) calculated in Eq. (2.15).

3.1 Position Measurement via Multilateration

The technique for obtaining the measured position of the receiver given the measured distances to each transmitter is similar to the trilateration method used by GPS discussed in Section 2.1.

First, circles centered at each transmitter are drawn with radii equal to the measured distance from the receiver to each transmitter. Zooming out of the room setup in Fig. 3.1, reveals these circles.

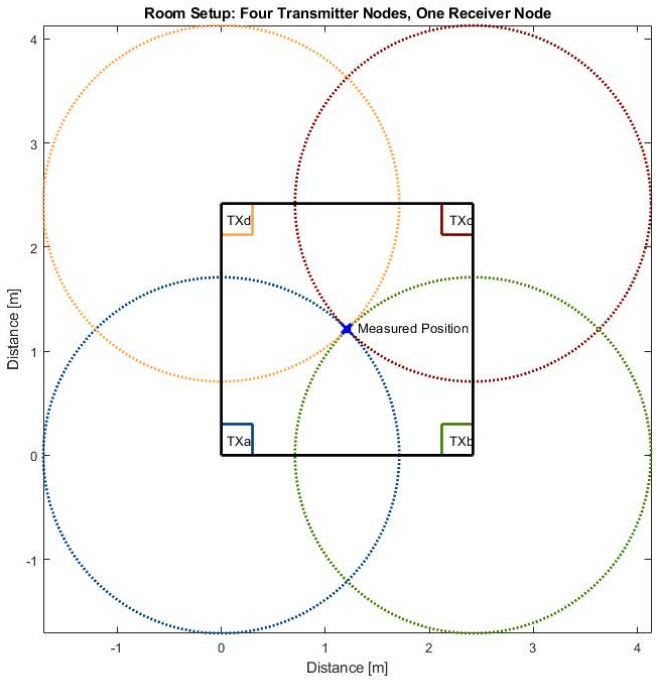

Figure 3.2: Multilateration Circles

The dimensions of the room are chosen such that the maximum measurable radius of each transmitter is greater than or equal to the diagonal of the room. This ensures that adjacent transmitters only intersect at one single point within the room. For example, transmitters TX_a (blue circle) and TX_b (green circle), intersect once outside of the room at about (1.3m, -0.8m) and once inside the room at the correct receiver position. The story is the same for transmitters TX_b and TX_c, TX_c and TX_d, and TX_d and TX_a.

Therefore, by measuring the four intersections of adjacent transmitters, four unique position measurements are obtained.

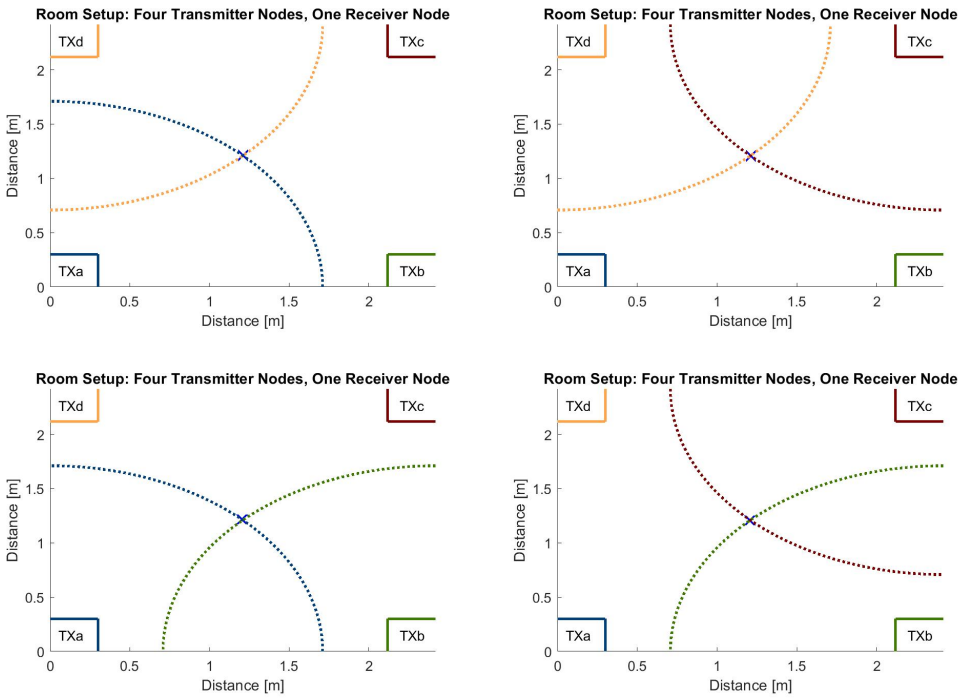

Figure 3.3: Intersections of Adjacent Transmitters' Multilateration Circles

These four position measurements are then averaged to produce one measurement. This averaging helps to eliminate error associated with the ultrasonic channel, the medium of travel, and random noise and jitter injected into the system.

This method is advantageous because it is easy to implement and relatively efficient in terms of computation time and power. The circles of adjacent transmitters intersect at two points, as seen in Fig. 2.1. To produce one unique position measurement, the intersection of all three transmitters must be calculated. This requires that the distance between each point in each circle be calculated. For K data points in the equation of a circle, this would be K^3 calculations. For the multilateration technique used in this thesis, $4 * K^2$ calculations are made to find the four intersections. Also, this multilateration method provides error-reduction unlike the single measurement produced by trilateration.

3.2 Ideal Simulation

First, the system is simulated as described in Section 2. The following table gives the characteristics of this simulation.

Table 3.1: Ideal Simulation Parameters

f_s	N	f_{IR}	λ_{12}	Tone Frequencies	Beat Frequency
1MHz	30000	100Hz	3.43m	a_1: 40kHz a_2: 40.1kHz b_1: 39.8kHz b_2: 39.9kHz c_1: 39.6kHz c_2: 39.7kHz d_1: 39.4kHz d_2: 39.5kHz	TX_a: 100Hz TX_b: 100Hz TX_c: 100Hz TX_d: 100Hz

3.2.1 Receiver-Transmitter Link Model

To begin the IPS simulation, the link between the receiver and the transmitter must be established. Fig. 3.4 shows a block diagram of this link for transmitter TX_a.

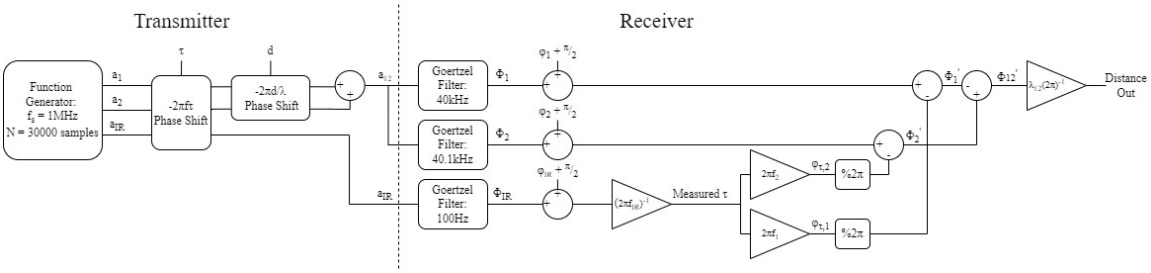

Figure 3.4: Block Diagram of Ideal Simulation of TX_a

At the transmitter side, the two sinusoidal ultrasonic tones and IR synchronization square wave signal must be synthesized using a high sampling rate f_s = 1MHz and block size N = 30000 samples. This ensures that the system captures 3 full beat periods, T_{12} = 10ms, of the two-tone signal. Then, all three signals undergo the same time delay, τ, which models the arbitrary start time of the waveform capture. The ultrasonic tones a_1 and a_2 undergo another delay that models the distance the waves have traveled. These two delays, distance and time, form the argument of the sine function in Eq. (2.6) without the phase shift, φ_{US}, imposed by the ultrasonic channel, meaning that they model the phases of the ideal tones. The synchronization signal does not have a delay associated with distance traveled because the signal transmission of infrared light is assumed to be instantaneous due to its high velocity (the speed of light). Next the two-tone signal a_{12} is produced by summing the two tones, thus completing the model of the transmitter.

The two-tone signal is processed by two Goertzel Filters tuned to the individual tone frequencies in order to measure the phases of the tones. The synchronization signal is processed by a third Goertzel Filter tuned

to the synchronization frequency. All three phases undergo a correction by constant $\varphi + \frac{\pi}{2}$ radians, which models the error associated with sampling a continuous signal. The $\frac{\pi}{2}$ reflects the fact that the Goertzel Filter calculates phase as if the input is a cosine wave, but the signals are sythesized as sine waves. In Section 3.3, these errors are caused by sampling and by real-world error such as φ_{US}, noise, and jitter. These constants are found experimentally by placing the receiver at known a distance, d, with known time shift, τ, and correcting the Goertzel Filters' output phases to match the expected phases for the known position. Using this method, the φ phase shifts are found:

Table 3.2: Phase Shifts, φ, for Ideal Simulation in Units of Radians

$\varphi_{\mathbf{a,IR}}$	$\varphi_{\mathbf{a,1}}$	$\varphi_{\mathbf{a,2}}$	$\varphi_{\mathbf{b,1}}$	$\varphi_{\mathbf{b,2}}$	$\varphi_{\mathbf{c,1}}$	$\varphi_{\mathbf{c,2}}$	$\varphi_{\mathbf{d,1}}$	$\varphi_{\mathbf{d,2}}$
$3.1416*10^{-4}$	0.252	0.252	0.2507	0.2507	0.2494	0.2494	0.2482	0.2482

Note that φ is equal for tones that belong to the same transmitter. This means that these φ values don't have an effect on the actual distance measurement because the values cancel out upon measuring the relative phase shifts of the two tones. These φ values only adjust the phases such that they display correctly in the simulation plots.

The phase output of the IR signal is used to calculate the perceived time delay of the sample set according to Eq. (2.17). This time delay is a measure of how much time has elapsed since the last zero-crossing of the ultrasonic signal, and it will always be less than or equal to 10ms (the beat period).

Using this measured time delay, the phases of the ultrasonic tones are synchronized in time by subtracting out the phase shift due to the time delay. These time phase shifts, $\varphi_{\tau,1}$ and $\varphi_{\tau,2}$, are calculated using Eq. (2.16) and then undergo a modulo by 2π block ensuring that they range from 0 to 2π. The modulo function returns the remainder after the division of the input by the specified divisor, in this case 2π. The resulting corrected phases, Φ_1' and Φ_2', are subtracted to yield the combined phase Φ_{12}', which is purely a function of distance, d.

Finally, the phase Φ_{12}' is transformed to a distance measurement via a linear transform which maps each phase measurement ranging from 0 to 2π radians into a unique distance measurement from 0 meters to the beat wavelength of the two-tone signal, in this case 3.43 meters.

This receiver-transmitter link design is repeated three times to create all four transmitters, TX_a, TX_b, TX_c, and TX_d, according to Table 3.1.

3.2.2 $\tau = 0$, Object Position = (0, 0)

The first simulation explored is with both τ and the object position set to 0. The object position input is translated into the input, d, in the simulation block diagram via calculating the distance between the object's coordinates and each respective transmitter.

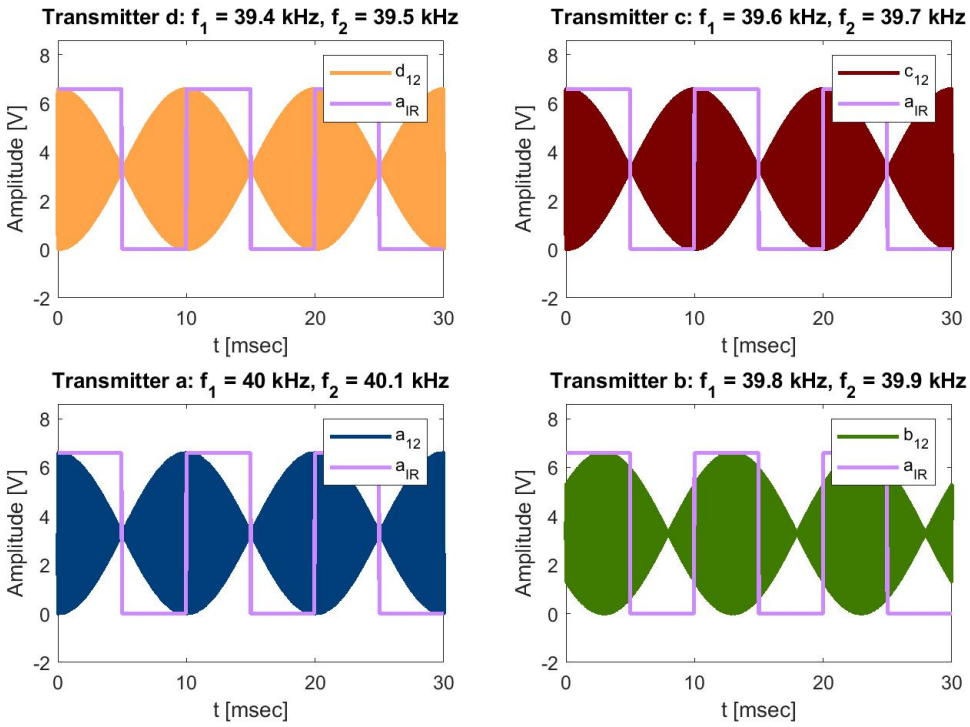

Figure 3.5: Received Signals, $\tau = 0$, Object Position = (0, 0)

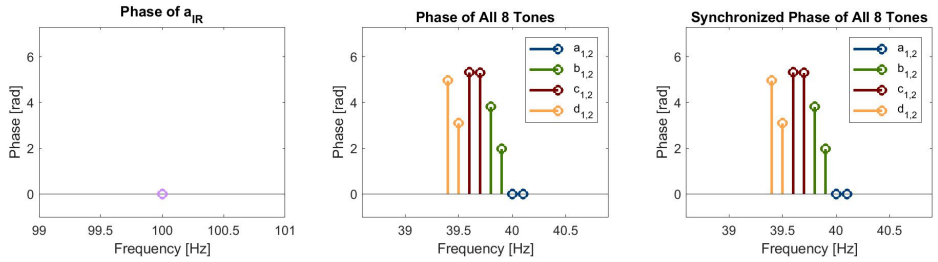

Figure 3.6: Signal Phases, $\tau = 0$, Object Position = (0, 0)

From Fig. 3.5 and Fig. 3.6, there is no phase shift in the IR signal, because τ is set to 0 and the IR signal is unaffected by distance. Also, observing the plot of the two-tone signal of TX_a, there is no phase shift because its distance to the receiver is 0. Note also from Fig. 3.6 that the measured and synchronized phases of the tones are identical because of the lack of time shift.

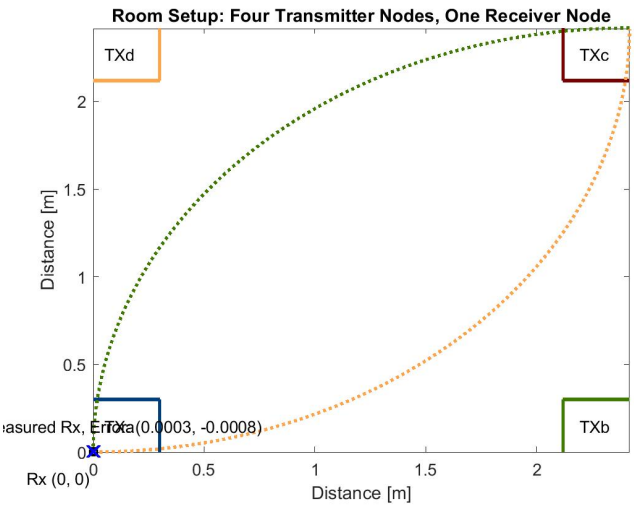

Figure 3.7: Position Measurement, $\tau = 0$, Object Position = (0, 0)

The actual and measured position of the receiver is then displayed within the room using the multilateration method. The TX_a and TXc circles are not visible because they are just outside the bounds of the plot.

3.2.3 $\tau = 0$, Object Position = ($\frac{W}{2}$ m, $\frac{L}{2}$ m)

Next, a simulation with the receiver in the center of the room is performed.

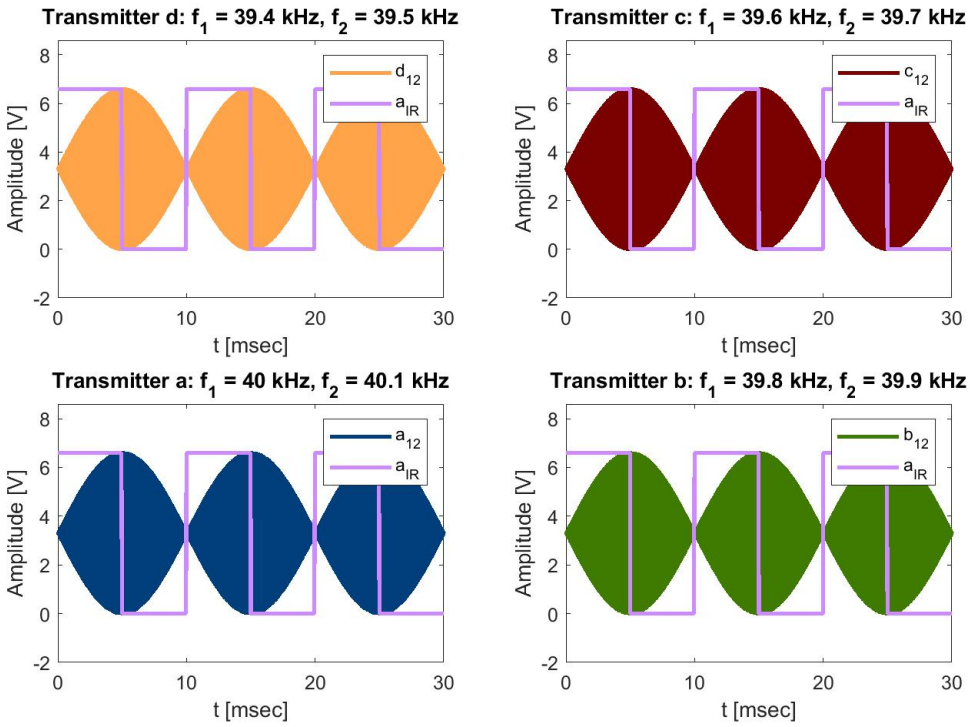

Figure 3.8: Received Signals, $\tau = 0$, Object Position = ($\frac{W}{2}$m, $\frac{L}{2}$m)

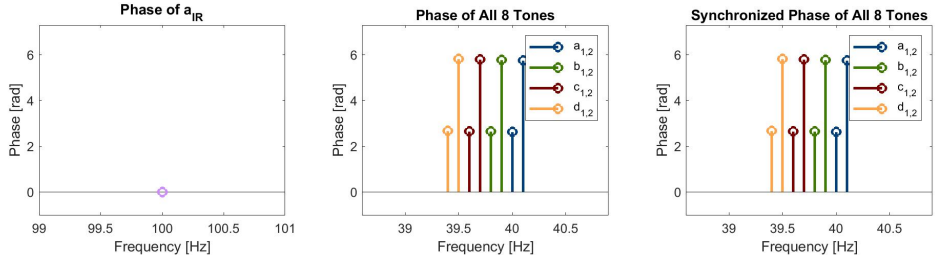

Figure 3.9: Signal Phases, $\tau = 0$, Object Position = ($\frac{W}{2}$m, $\frac{L}{2}$m)

The phase of the IR synchronization signal stays at 0 because τ remains at 0. However, the two-tone ultrasonic signals from all of the transmitters have a relative phase shift of π radians, as they are measuring half of their maximum distances.

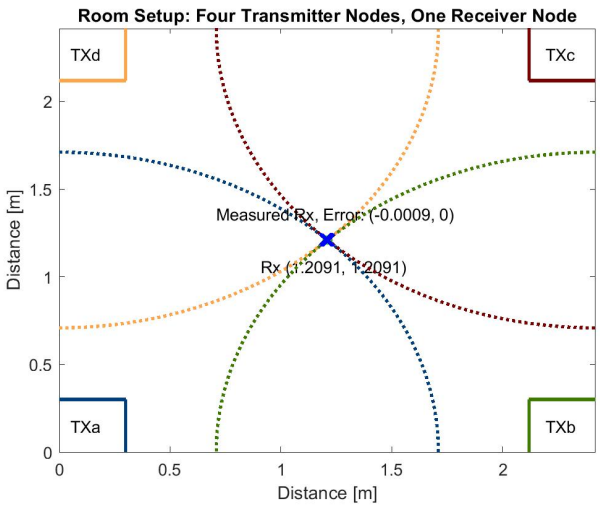

Figure 3.10: Position Measurement, $\tau = 0$, Object Position = ($\frac{W}{2}$m, $\frac{L}{2}$m)

This is shown in the preceding plot, where the receiver is exactly in the middle of the room.

3.2.4 τ = 5ms, Object Position = (0, 0)

After two tests of distance phase shifts, a time phase shift test is performed by setting τ = 5ms, or half of the beat period.

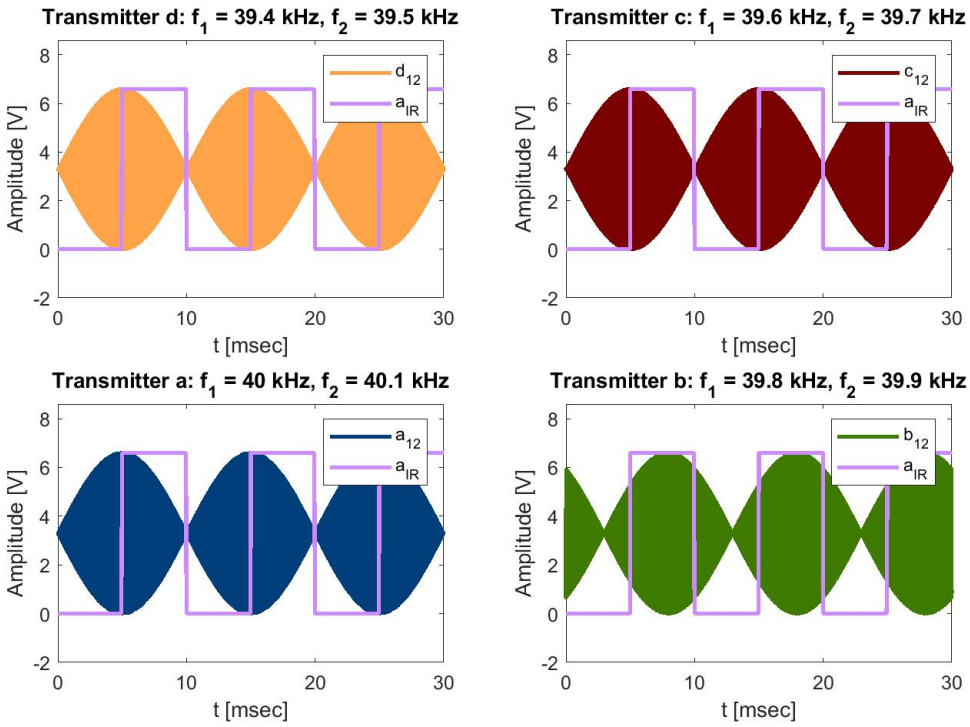

Figure 3.11: Received Signals, τ = 5ms, Object Position = (0, 0)

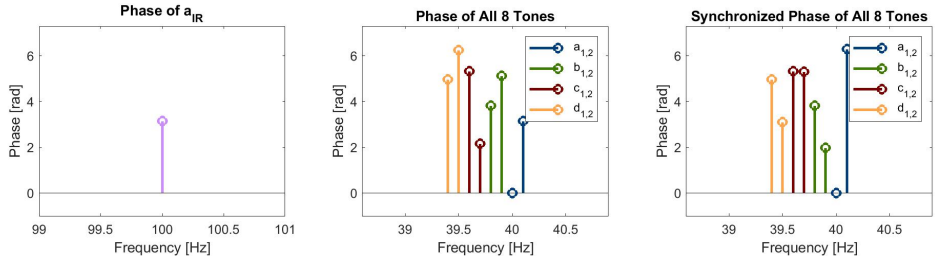

Figure 3.12: Signal Phases, τ = 5ms, Object Position = (0, 0)

Now, the figures show an observed phase shift of π radians for the IR signal because the time delay is half of its period. The two-tone signals also exhibit a phase shift of π radians since their beat period is equal to the IR signal's period.

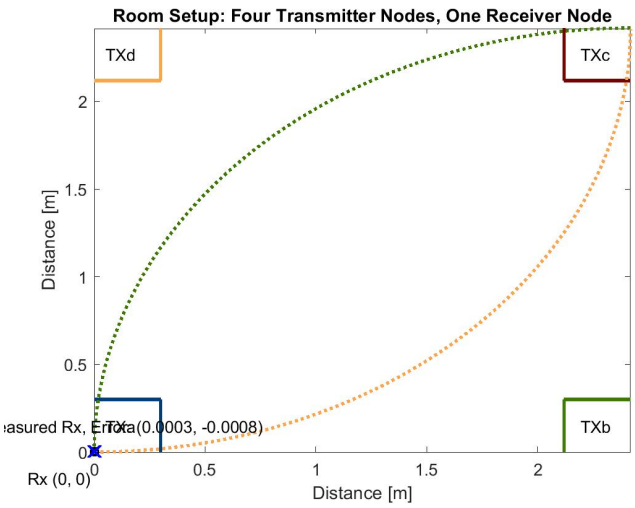

Figure 3.13: Position Measurement, τ = 5ms, Object Position = (0, 0)

However, there is no change in the object's measured position between this simulation and the simulation in Section 3.2.2, suggesting the system's time synchronization is functioning properly.

3.2.5 τ = Random, Object Position = (Random, Random)

Finally, a simulation is performed with random values of τ and the object's position. τ is a uniformly distributed variable on the interval [0:10ms] and W and L are independent uniformly distributed random variables on the interval [0:2.4183m]. These variables are $\tau = 8.212$ms and Object Position = (0.5678m, 0.8540m).

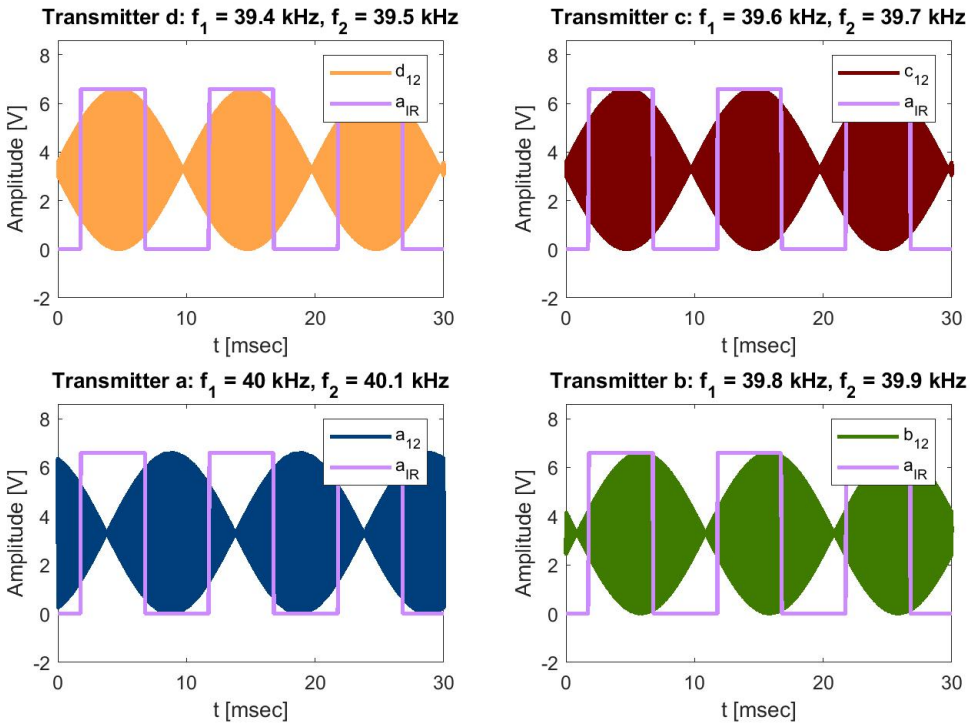

Figure 3.14: Received Signals, τ = 8.212ms, Object Position = (0.5678m, 0.8540m)

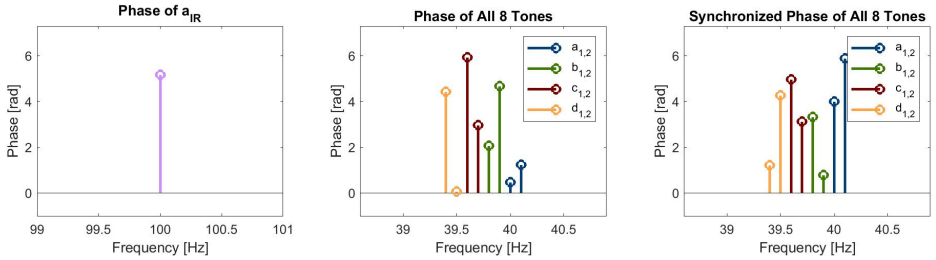

Figure 3.15: Signal Phases, τ = 8.212ms, Object Position = (0.5678m, 0.8540m)

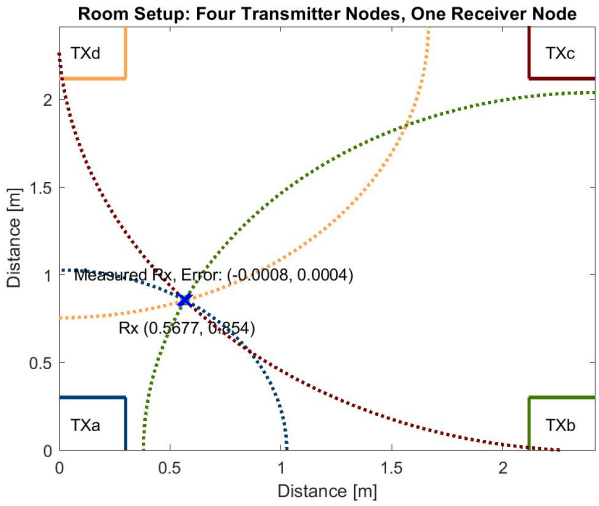

Figure 3.16: Position Measurement, τ = 8.212ms, Object Position = (0.5678m, 0.8540m)

3.3 System Simulation

Now that the ideal system has been proven in simulation, the real-world system simulation is performed using the parameters of the hardware in Section 4. The sampling rate for the ADC onboard the MSP432P401R microctroller was found to be approximately 101.1kHz, slightly above the Nyquist sampling rate of 80kHz for a 40kHz-based system [25]. The tones were found to be not perfectly orthogonal, as seen by the non-constant tone spacing in Table 3.3. The reason for this imperfect orthogonality is related to the hardware and is discussed further in Section 4.2.2. Both of these non-idealities manifest themselves as deterministic error which is corrected for in the simulation using φ. However, as discussed in Section 4, the issue of non-orthogonality is more pronounced in the hardware implementation.

The same four test setups are used: 0 time, 0 position; 0 time, middle position; half-beat period, 0 position; random time, random position.

Table 3.3: System Simulation Parameters

f_s	N	f_{IR}	λ_{12}	Tone Frequencies	Beat Frequency
101.1kHz	3152	96.231Hz	3.5643m	a_1: 40kHz a_2: 40.09623kHz b_1: 39.8089kHz b_2: 39.9042kHz c_1: 39.5883kHz c_2: 39.6833kHz d_1: 39.4011kHz d_2: 39.4945kHz	TX_a: 96.231Hz TX_b: 95.313Hz TX_c: 94.258Hz TX_d: 93.366Hz

Because the beat wavelength, λ_{12}, has changed compared to the ideal simulation, the room size must change too. The new beat wavelength is 3.5643m calculated using Eq. (2.3), so the new (W,L) becomes (2.5133, 2.5133) where:

$$\text{Diagonal} = \sqrt{\mathrm{W}^2 + \mathrm{L}^2} = \sqrt{(2)(2.5133^2)} = 3.5543\text{m} \tag{3.2}$$

Using the same method as in Section 3.2, the new φ values are found:

Table 3.4: Phase Shifts, φ, for Ideal Simulation in Units of Radians

$\varphi_{a,IR}$	$\varphi_{a,1}$	$\varphi_{a,2}$	$\varphi_{b,1}$	$\varphi_{b,2}$	$\varphi_{c,1}$	$\varphi_{c,2}$	$\varphi_{d,1}$	$\varphi_{d,2}$
$1.4946*10^{-2}$	0.9838	0.9858	0.8553	0.9579	0.4859	0.7067	0.0046	0.3021

3.3.1 Receiver-Transmitter Link

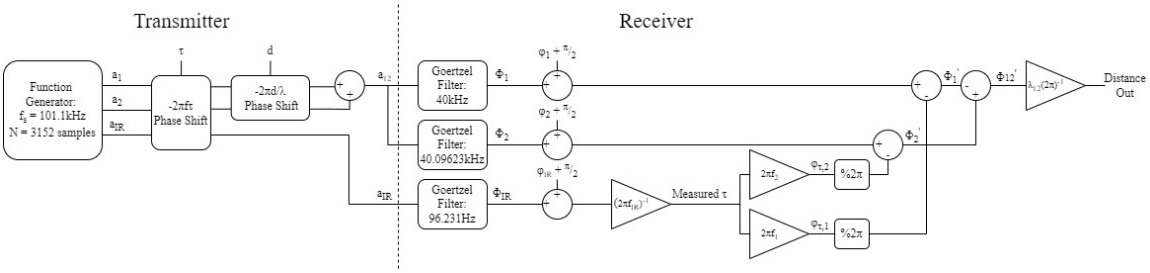

Figure 3.17: Block Diagram of System Simulation of TX_a

The block diagram of the system simulation of TX_a is shown in Fig. 3.17. The only difference between it and the ideal simulation's block diagram are the parameters in Table 3.3 and Table 3.4.

3.3.2 $\tau = 0$, Object Position = (0, 0)

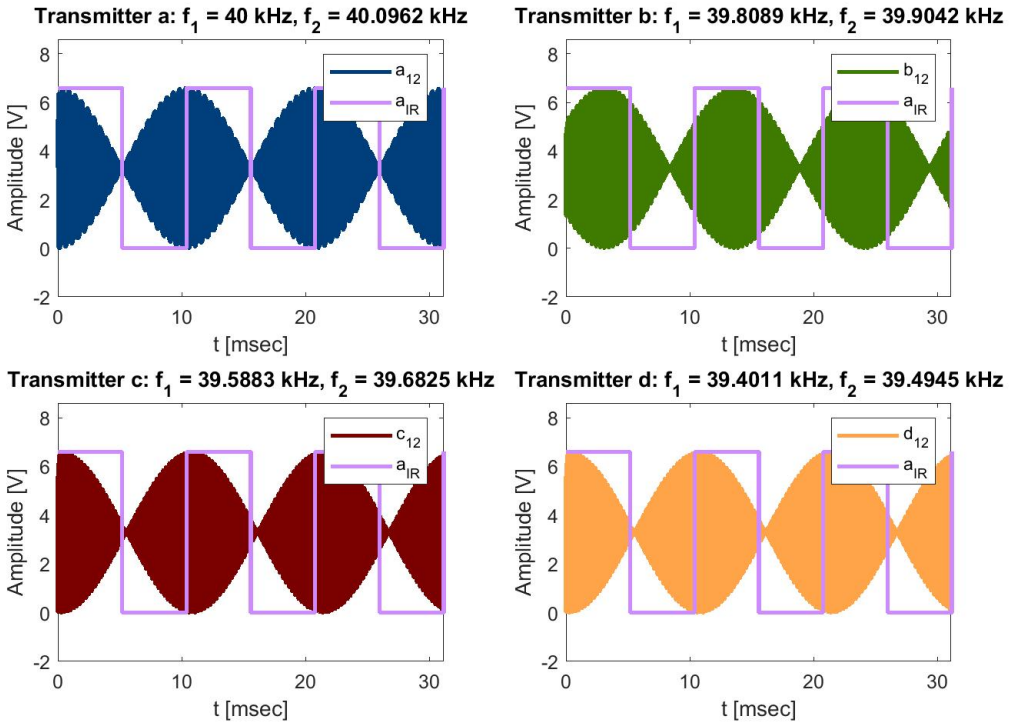

Figure 3.18: Received Signals, $\tau = 0$, Object Position = (0, 0)

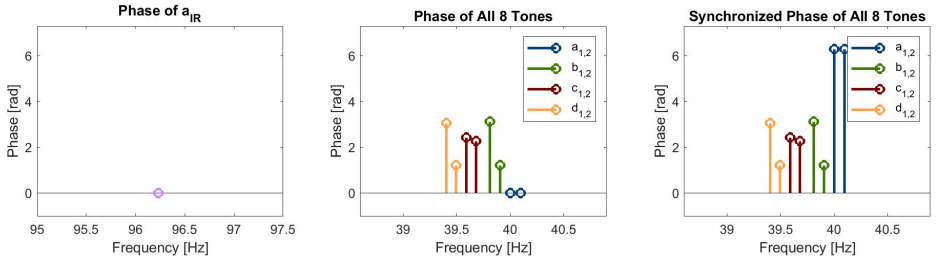

Figure 3.19: Signal Phases, $\tau = 0$, Object Position = (0, 0)

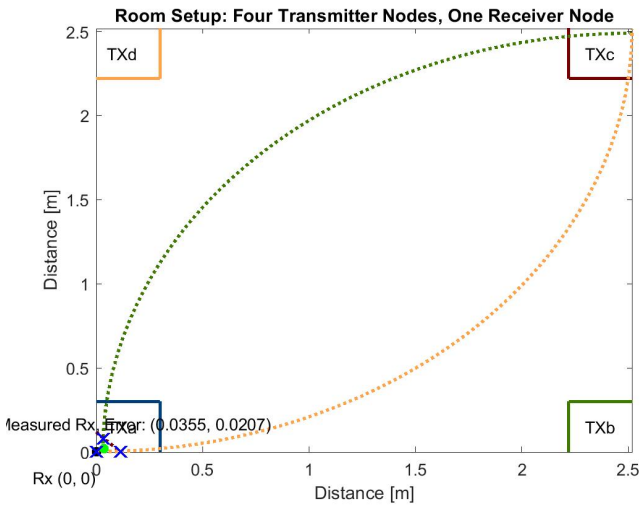

Figure 3.20: Position Measurement, $\tau = 0$, Object Position = (0, 0)

3.3.3 $\tau = 0$, Object Position = ($\frac{W}{2}$m, $\frac{L}{2}$m)

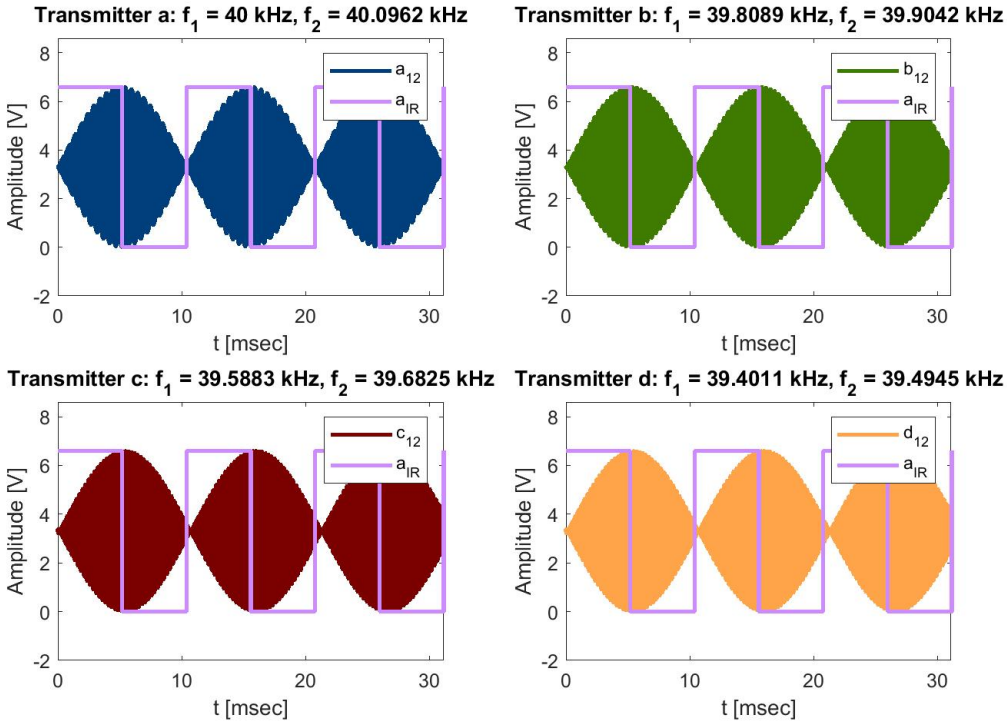

Figure 3.21: Received Signals, $\tau = 0$, Object Position = ($\frac{W}{2}$m, $\frac{L}{2}$m)

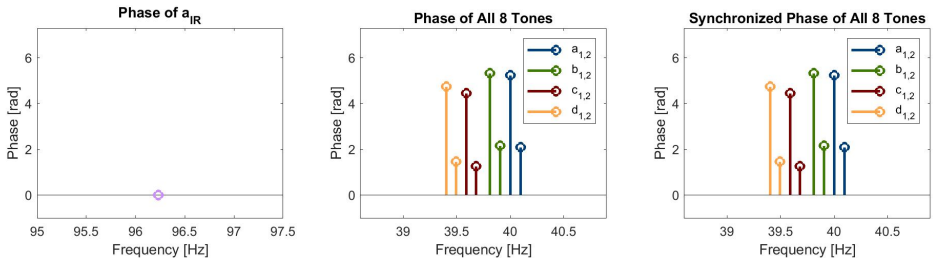

Figure 3.22: Signal Phases, $\tau = 0$, Object Position = ($\frac{W}{2}$m, $\frac{L}{2}$m)

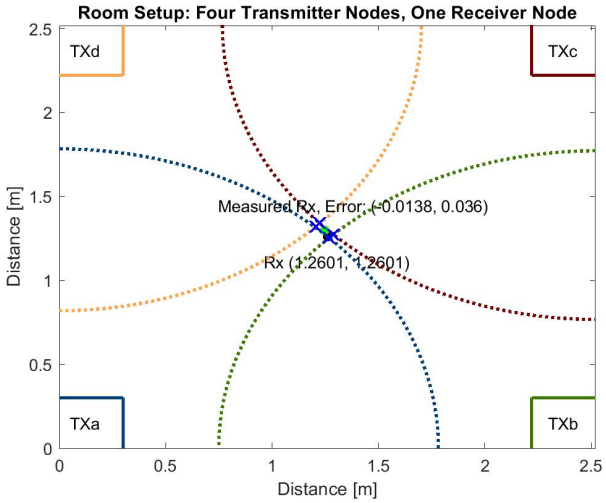

Figure 3.23: Position Measurement, $\tau = 0$, Object Position = ($\frac{W}{2}$m, $\frac{L}{2}$m)

3.3.4 τ = 5.19583ms, Object Position = (0, 0)

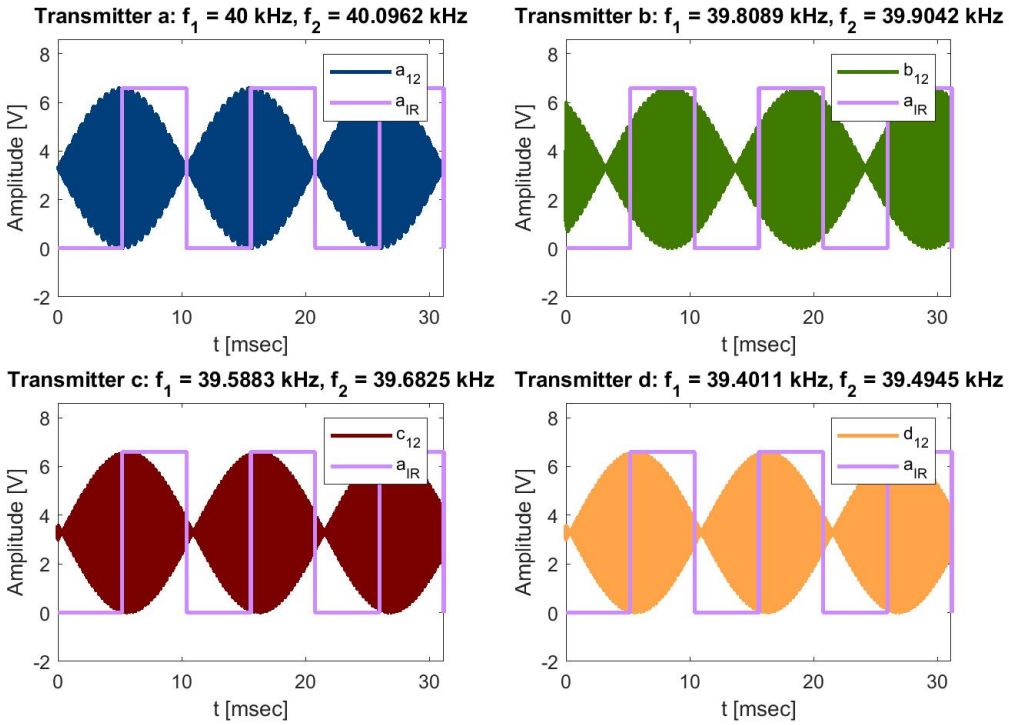

Figure 3.24: Received Signals, τ = 5.19583ms, Object Position = (0, 0)

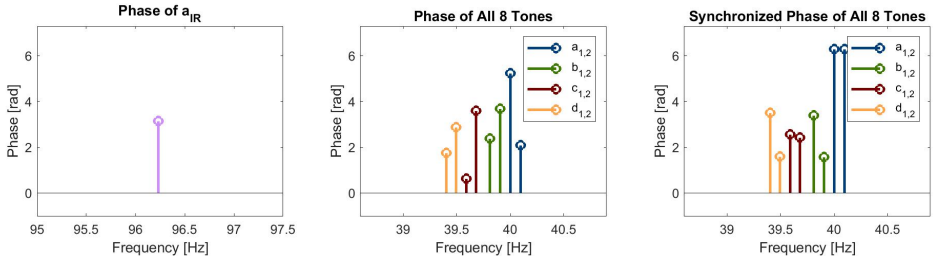

Figure 3.25: Signal Phases, τ = 5.19583ms, Object Position = (0, 0)

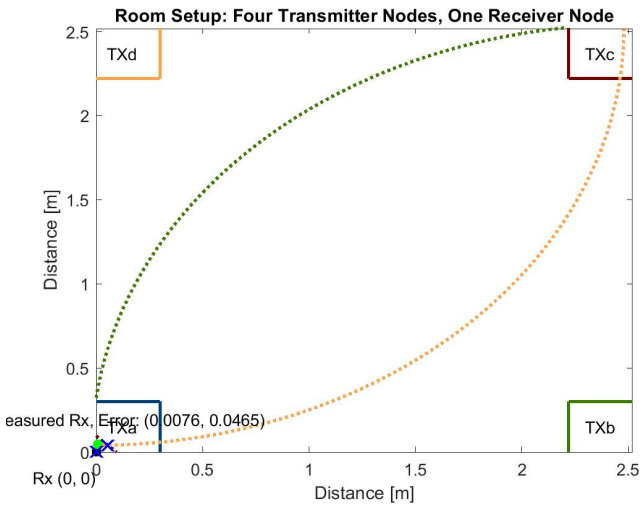

Figure 3.26: Position Measurement, τ = 5.19583ms, Object Position = (0, 0)

3.3.5 τ = Random, Object Position = (Random, Random)

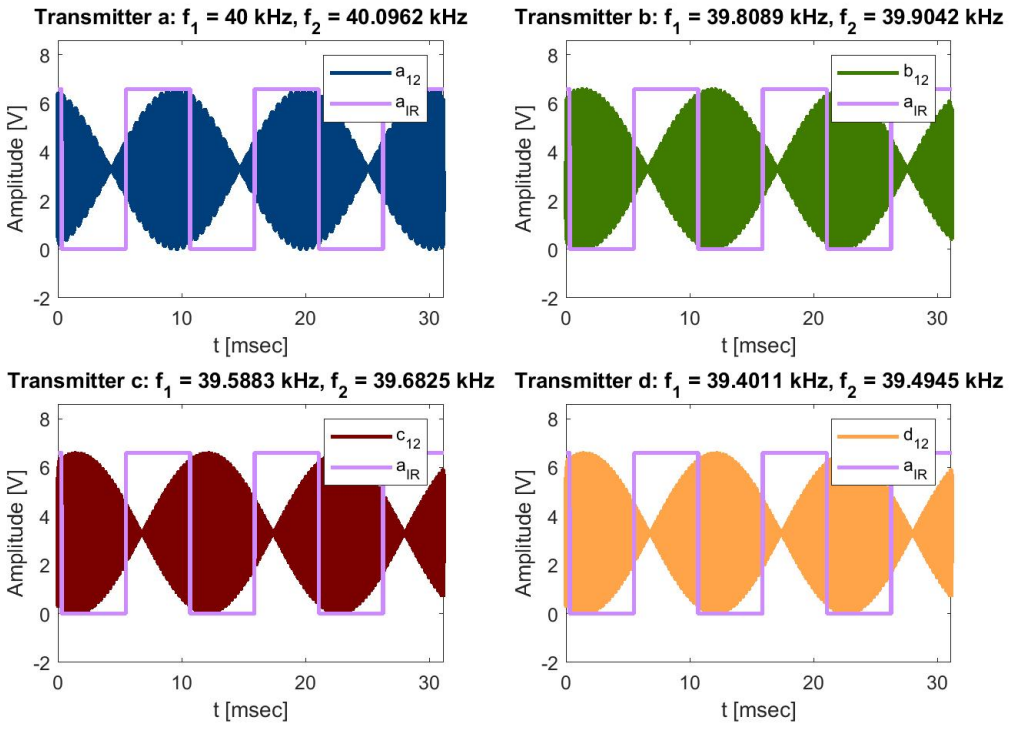

Figure 3.27: Received Signals, τ = 4.89539ms, Object Position = (1.7963m, 1.2614m)

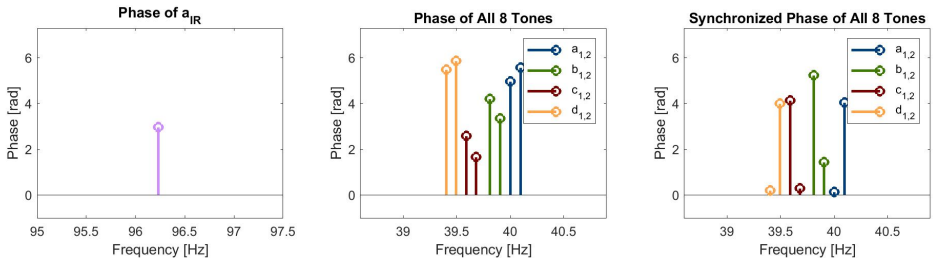

Figure 3.28: Signal Phases, τ = 4.89539ms, Object Position = (1.7963m, 1.2614m)

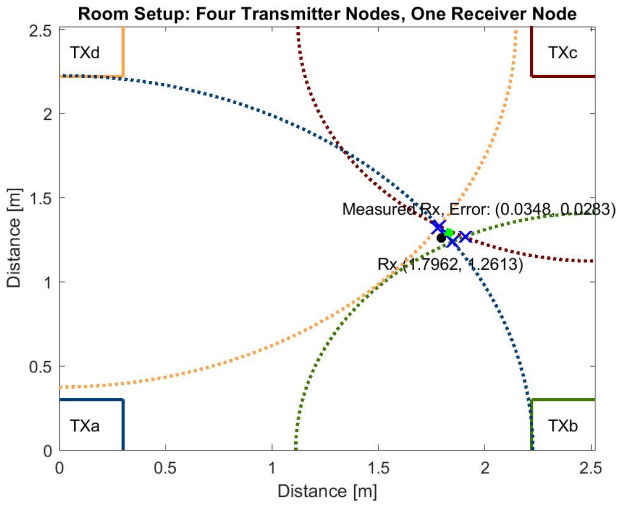

Figure 3.29: Position Measurement, τ = 4.89539ms, Object Position = (1.7963m, 1.2614m)

The preceding four simulations show that the real-world system modeled by MATLAB yields favorable results, where the error is only slightly higher in magnitude than that of the ideal simulation. This proves true for small block size, N, but the error grows larger in magnitude as N increases because the signal phases begin to drift apart. This phenomenon is explained further in the next section.

4 Implementation

As mentioned in Section 1.2, the hardware implementation of this design explored in this section consists of the link between one transmitter with IR synchronization, TX_a, and the receiver. The concepts here can easily be repeated three times to create the full IPS, but such an attempt is beyond the scope of this thesis. The Verilog code used to program the transmitter and the C code used to program the receiver can be found in the Appendix.

4.1 Transmitter TX_a

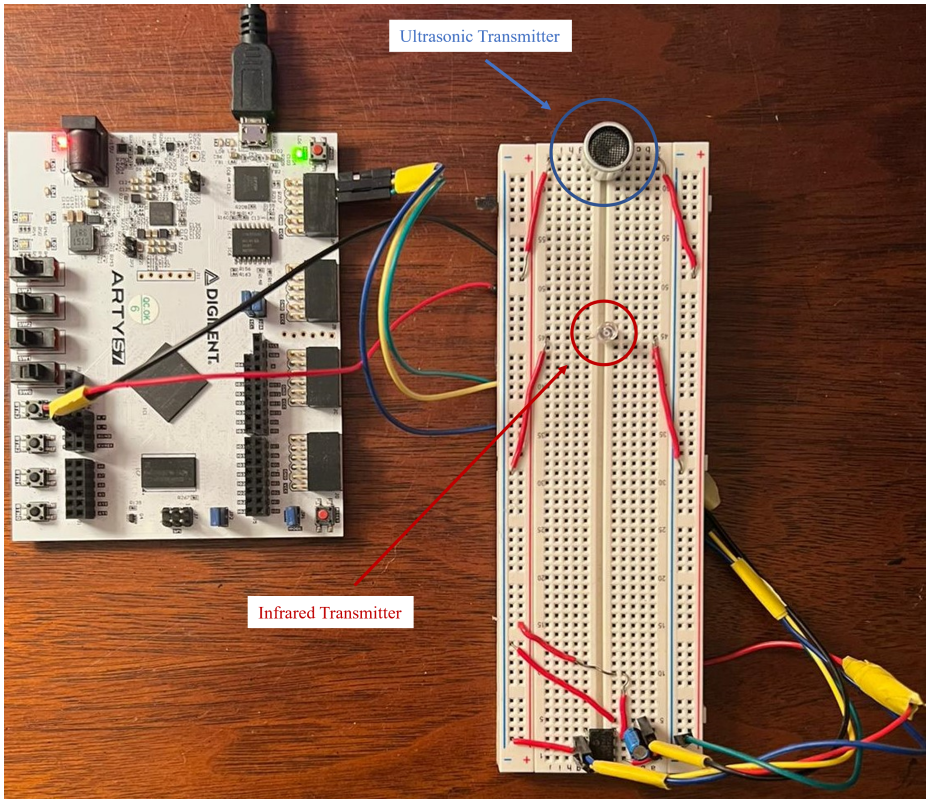

Figure 4.1: Photograph of Transmitter Node

The first subsystem explained is the transmitter node, TX_a, with real tone frequencies of 40kHz and 40.09623kHz and IR synchronization frequency of 96.231Hz. The 40T12B-R ultrasonic transmitter [14] and EK8443 IR diode [27] are chosen because they are inexpensive and widely available. All signals are generated by an Arty S7 FPGA development board [26] powered and programmed by a laptop computer.

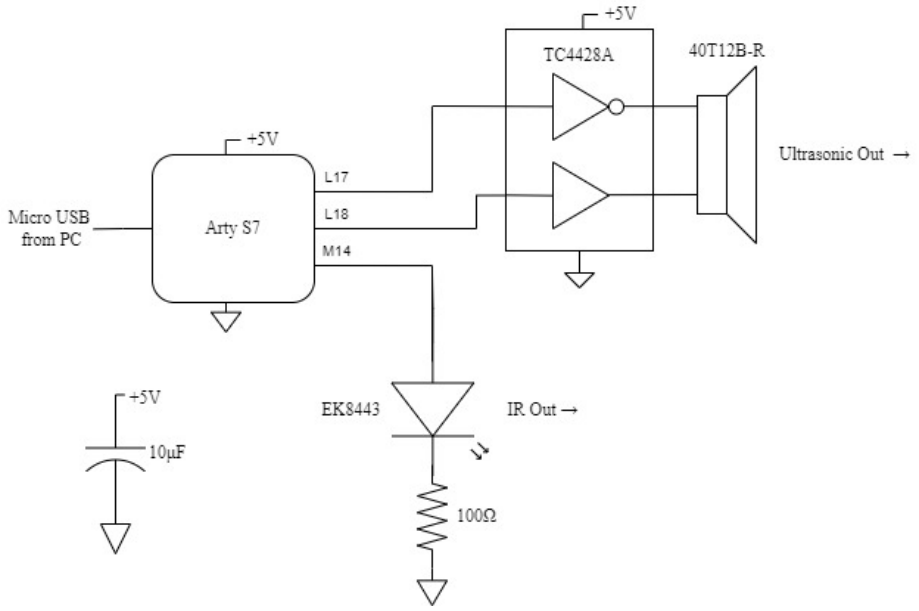

Figure 4.2: Schematic of Transmitter Node

The block diagram of the transmitter node is shown in Fig. 4.2. The L17, L18, and M14 pins of the FPGA belong to the PMOD header JA on the Arty S7 board. The waveforms a_1 and a_2 are output through L17 and L18. Note that at this stage in the design, a_1 and a_2 are square waves with fundamental frequencies equal to the desired tone frequencies. These desired tone frequencies are then extracted by the bandpass characteristics of the ultrasonic channel, leaving only the fundamental tone at the receiver side. This effect is discussed in Section 4.1.1.

The square waves are fed into the two inputs of the TC4428A 1.5A Dual High-Speed Power MOSFET Drivers [28] whose outputs are connected across the terminals of the ultrasonic receiver. The Arty S7 board produces the +5V supply voltage to the TC4428A. This setup creates the high-power, square wave version of the two-tone signal a_{12} at the input of the ultrasonic receiver, which gets converted into the sum of the desired sinusoidal tones by the ultrasonic channel.

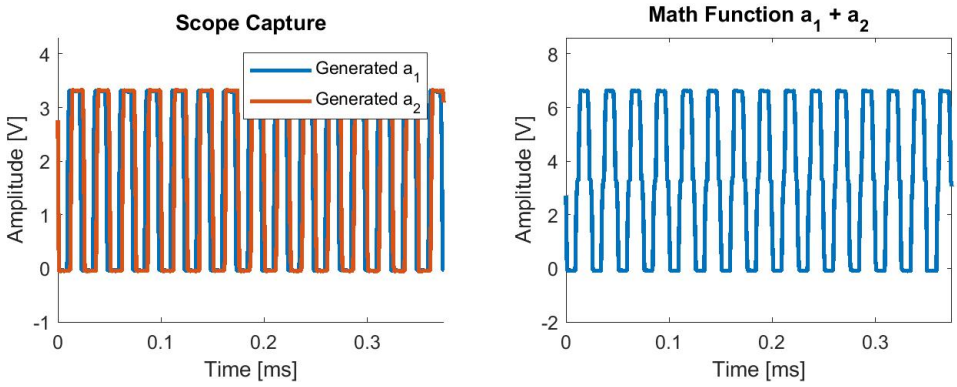

Figure 4.3: Square Waves of a_1 and a_2 as Created by the FPGA and Measured by the AD2; Square Wave a_{12} as Created by Addition of Measured Square Waves of a_1 and a_2 in MATLAB

Fig. 4.3 and Fig. 4.4 show these signals at the output of the FPGA. An Analog Discovery 2 (AD2) USB Oscilloscope [29] is used to capture the two waveforms, and then the data is exported to MATLAB in Comma Separated Value (CSV) files for plotting. The right plot shows the result of adding the measured square waves of a_1 and a_2 in MATLAB, creating the square wave version of a_{12}. During the time period of this capture, the square waves are in alignment, or nearly zero phase, and are constructively interfering. However, just as in the plots of the two-tone signals in the simulations in Section 3, these square waves destructively interfere once every beat period, creating the nulls seen in Fig. 4.4 (circled in red).

This summed square wave represents the signal that the ultrasonic transducer sees. By Kirchoff's Voltage Law (KVL), the voltage across the terminals of the transducer will be the difference of the outputs of the TC4482A IC. Since a_1 is inverted in the TC4482A and the ultrasonic transmitter is bidirectional, the voltage seen by the transducer is $|a_1 + a_2|$. This, in conjunction with the bandpass channel that selects only the fundamental frequencies of the square waves, is highly favorable as it provides an easy method of transmitting a strong, sinusoidal two-tone signal using an inexpensive, nonlinear, and low-power amplifier with a simple switching input.

Extending the time axis on the plot of the square wave a_{12} in Fig. 4.3 reveals its beat period of about $\frac{1}{96.231\text{Hz}}$ = 10.039ms in Fig. 4.4. Since this plot was created by adding the two measured square waves in MATLAB and not measured at the terminals of the ultrasonic transmitter, the display peak-to-peak voltage is the sum of the logic high voltages of a_1 and a_2, $3.3V_{PP} + 3.3V_{PP} = 6.6V_{PP}$. However, since the TC4428A uses a 5V supply, effectively converting the logic high voltage to 5V, the peak-to-peak voltage that the transducer will see is $10V_{PP}$. This is well within the operating range of the device as defined by its datasheet ($<30V_{PP}$).

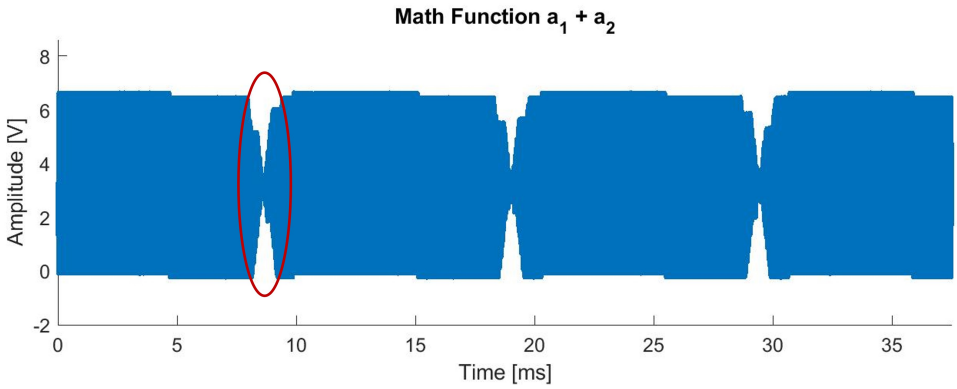

Figure 4.4: Extended Square Wave a_{12} as Created by MATLAB

Fig. 4.5 shows the null zone of the signal encircled in red in the Fig. 4.4. This is the destructive interference that creates the beat period of the ultrasonic signal.

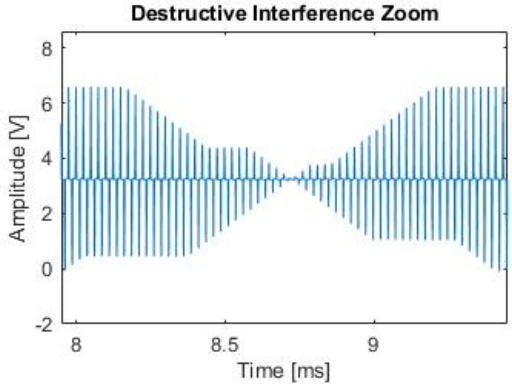

Figure 4.5: Zoom of Destructive Interference in Square Wave Version of a_{12}

Pin M14 of the FPGA drives the IR synchronization signal via a square wave with 50% duty cycle and frequency equal to the beat frequency of the two-tone ultrasonic signal. As calculated in Eq. (4.3), this frequency is f_{IR} = 96.231Hz.

From the IR LED's data sheet, the forward voltage of the diode and its maximum current rating are approximately 1.2V and 30mA, respectively. Using the standard 3.3V logic level output of the FPGA PMOD pins, the maximum current through the photodiode is 21mA per the following:

$$I_{max} = \frac{(3.3V - 1.2V)}{100\Omega} = 21mA \tag{4.1}$$

4.1.1 Ultrasonic Channel

In this section, the characteristics of the ultrasonic transducers are analyzed to select the best tone frequencies. The transducers chosen are the 40T/R12B-R for their affordable price and wide availability. The data is gathered from work performed in a previous class, EE 449 Electronic Design Laboratory, in which the same transducers were used. [30] [31]

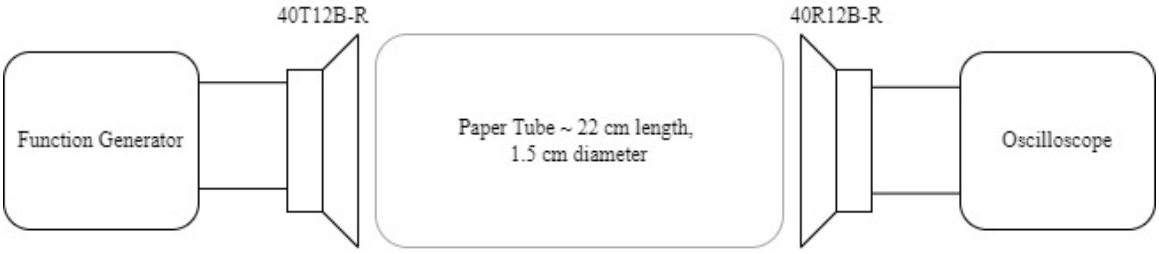

Figure 4.6: Test Setup for Obtaining Frequency Response of Ultrasonic Channel

The following figures show the impedance of the transmitter and the frequency response of the channel accounting for both the ultrasonic transmitter and receiver. These measurements is obtained by sweeping input frequencies at the transmitter and measuring the response of the receiver using a paper tube to isolate the transducers, shown in Fig. 4.6.

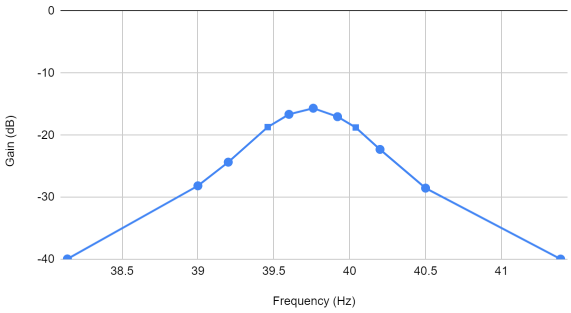

Figure 4.7: Frequency Response of Ultrasonic Channel [30]

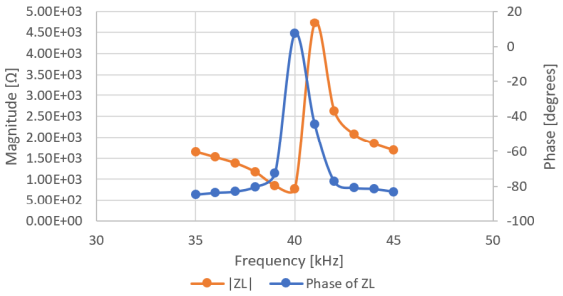

Figure 4.8: Impedance of Ultrasonic Transmitter [30]

The channel is bandpass in nature with a bandwidth of about 700Hz and 3dB points at 39.4kHz and 40.1kHz. This works favorably, as it smooths the combined square wave input to the transmitter into a two-tone sinusoidal signal at the receiver by eliminating the harmonics of the square waves, leaving only sinusoids at the fundamental frequencies.

The magnitude of the impedance of the transmitter is lowest at about 40kHz, suggesting the highest transfer of

electrical energy to acoustic energy occurs at that frequency. The plot also displays the phase of the ultrasonic transmitter's impedance. This phase shift is one of the contributors to φ_{US}, as discussed in Section 2.6.1, along with phase shifts associated with propagation through the medium of travel (air).

Table 4.1: Tone Frequencies

Transmitter	$\mathbf{f_1}$	$\mathbf{f_2}$	$\mathbf{f_{IR}}$
TX_a	40kHz	40.1kHz	100Hz
TX_b	39.8kHz	39.9kHz	-
TX_c	39.6kHz	39.7kHz	-
TX_d	39.4kHz	39.5kHz	-

Given the ultrasonic channel's characteristics and the desired beat frequency of 100Hz as stated in Section 2, the most ideal selection of tone frequencies is shown in Table 3.1.

4.1.2 Real Tone Frequencies

The tone frequencies presented in Table 4.1 are ideal and perfectly spaced at 100Hz. However, late in the implementation of TX_a, this thesis found that these exact tone frequencies could not be achieved with the Arty S7 board [26]. This is due to the limitation imposed by the board's clock frequency on its ability to generate square waves. The signals are generated by counting clock cycles up to defined constants and changing the state of an output when its respective counter is full. For the maximum allowable onboard clock frequency of 100MHz, the equation to calculate the generated 50% duty cycle square wave's frequency is:

$$f_{tone} = 0.5 * \frac{f_{CLK}}{Counter} \tag{4.2}$$

And, rounded for viewing's sake, the tones become:

Table 4.2: Tone Frequencies

Transmitter	Counter 1	$\mathbf{f_1}$	Counter 2	$\mathbf{f_2}$	$\mathbf{f_{12}}$
$\mathbf{TX_a}$	1250	40kHz	1247	40.0962kHz	96.231Hz
$\mathbf{TX_b}$	1256	39.8089kHz	1253	39.9042kHz	95.313Hz
$\mathbf{TX_c}$	1263	39.5883kHz	1260	39.6833kHz	94.258Hz
$\mathbf{TX_d}$	1269	39.4011kHz	1266	39.4945kHz	93.366Hz

The IR synchronization signal is victim to the same time-discretization error imposed by the device's clock. Since TX_a is built in hardware, its beat frequency is used as the IR synchronization frequency. The counter

value found to produce a square wave with frequency closest to the beat period of TX_a is 519,583.

$$f_{IR} = 0.5 * \frac{100\text{MHz}}{519,583} = 96.231\text{Hz} \tag{4.3}$$

Table 4.3 shows the error in IR synchronization frequency for each transmitter. The error is very small, but not 0, for TX_a because the synchronization frequency is determined by TX_a. It is much larger for all other transmitters.

Table 4.3: IR Synchronization Frequency Error

TX_a	**TX_b**	**TX_b**	**TX_b**
$6.1736 * 10^{-5}$ Hz	0.9184Hz	1.9732Hz	2.8635Hz
6.4154*10^{-5} %	0.95%	2.05%	2.98%

4.2 Receiver

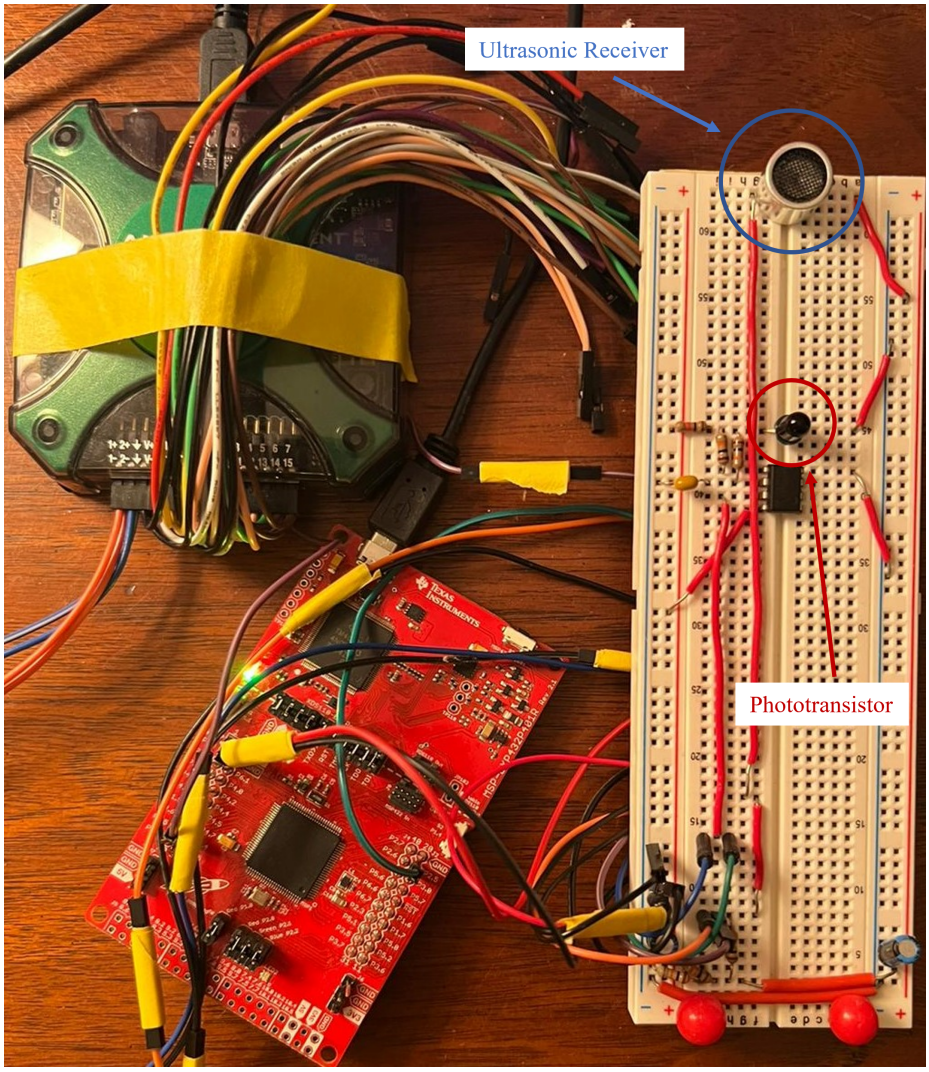

Figure 4.9: Photograph of Receiver Node

The receiver node contains a 40R12B-R ultrasonic receiver [14] and EK8443 phototransistor [27], the complementary receivers to the inexpensive and readily available transmitters. After some conditioning, the ultrasonic signal from the transmitter is sampled using an ADC and processed using a computer, and the IR signal is captured by checking the input on a digital pin for logic high or logic low. Two different ADC and computer setups are tested. The first setup uses the ADC of the Analog Discovery 2 to sample the signal then uses MATLAB to process the exported data. The second consists solely of the MSP432P401R (MSP432) microcontroller [33] with onboard ADC, in use for both sampling and processing. The latter setup is the ultimate goal of the system, since it allows for a fully embedded receiver rather than one that requires a laptop for signal processing.

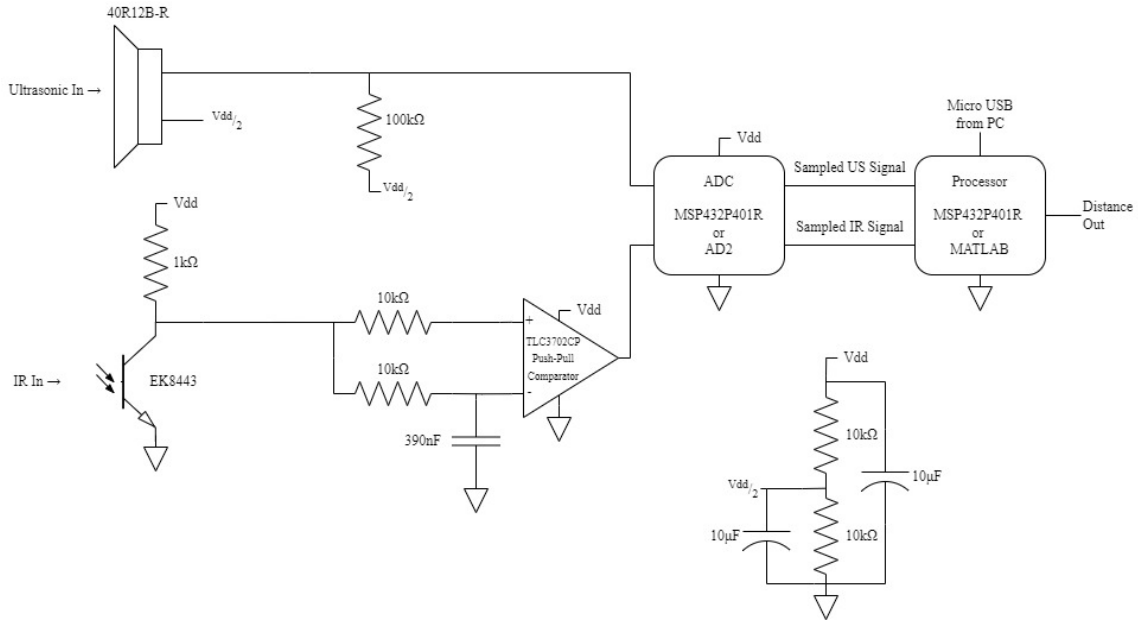

Figure 4.10: Schematic of Receiver

Fig. 4.10 shows the general receiver schematic. The ultrasonic receiver operates at half-supply via the capacitor-bypassed, resistive divider in the bottom right of the image. Without this DC biasing, the received ultrasonic waveform would have an DC value of 0V, meaning half of the waveform would be negative valued. With a reference voltage of 0V, the MSP432's ADC requires an operating range of 0-3.3V [33]. Therefore, using V_{dd} = 3.3V sourced from the MPS432, the transducer must be biased at approximately 1.65V. Using V_{dd} = 5V sourced from the AD2, the transducer is biased at roughly 2.5V. The output of the transducer is then sampled and processed by the ADC and processor.

The IR receiver device is a phototransistor, meaning that light incident on the device induces a small current at its base, allowing current proportional to this base current to flow from the collector through the emitter. Thus, a 1kΩ pull-up resistor tied to the supply voltage of the microctrontroller board (MSP432 or AD2) is used to induce a voltage at the collector node of the device. This technique inverts the input IR signal, since the phototransistor pulls this node down to ground when it senses light (input high) and the resistor pulls the node up to the supply when the device senses no light (input low). This corresponds to a π phase shift of the received IR signal, which must be accounted for within φ_{IR}. This effect is demonstrated in Fig. 4.11.

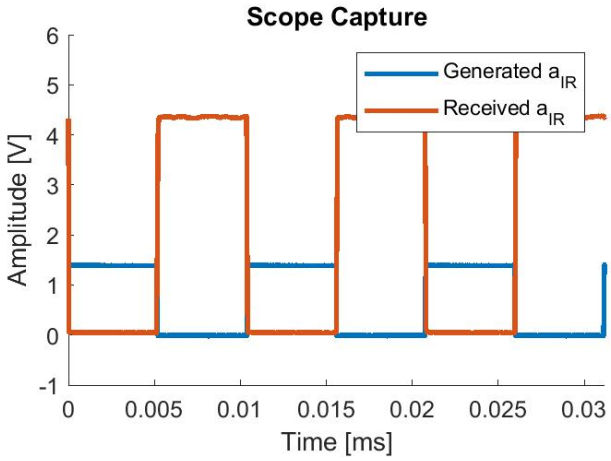

Figure 4.11: Scope Capture of Generated IR Signal and Received IR Signal at the Output of The TLC3702CP Comparator

From the scope capture, there is an apparent inversion between the generated (blue) waveform and the received (red) waveform, resulting in a π phase shift of the fundamental. The generated a_{IR} shows has an approximate amplitude of $1.2V_{PP}$, which is consistent with the current calculation in Eq. (4.1).

The EK8443 phototransistor is sensitive to all light, not just IR light. For this reason, it has a constant DC bias due to the ambient light surrounding the device. The two 10kΩ resistors and 390nF bypass capacitor form a circuit that allows the TLC3702CP [34] to compare the IR signal with its DC point. This effectively measures only the AC component of the received signal, where the AC component is the received square wave emitted by the IR transmitter.

4.2.1 Verification of Design with AD2 and MATLAB

First, the system is implemented using the data capture feature of the Analog Discovery 2 (AD2) set to f_s = 1MHz and N = 31175. This captures approximately 3.000002 periods of the IR synchronization signal, exactly 1247 periods of the 40kHz tone, exactly 1250 periods of the 40.09623kHz tone, and exactly 3 beat periods of the two-tone signal. These are calculated with the following equations:

$$\text{Number of Periods in IR Signal} = \frac{31175 * f_{IR}}{f_s} = \frac{31175 * 96.231\text{Hz}}{1\text{MHz}} = 3.000002 \text{ Periods} \tag{4.4}$$

$$\text{Number of Periods in } a_1 = \frac{31175 * 40\text{kHz}}{1\text{MHz}} = 1250 \text{ Periods} \tag{4.5}$$

$$\text{Number of Periods in } a_2 = \frac{31175 * 40.09623095\text{kHz}}{1\text{MHz}} = 1247 \text{ Periods} \tag{4.6}$$

$$\text{Number of Periods in } a_{12} = \frac{31175 * 96.23095\text{Hz}}{1\text{MHz}} = 3 \text{ Periods} \tag{4.7}$$

The block diagram for this implementation is seen in Fig. 4.12. The signals are sampled by the AD2 through its two probe channels, and the data is then exported into MATLAB on a laptop computer via CSV files.

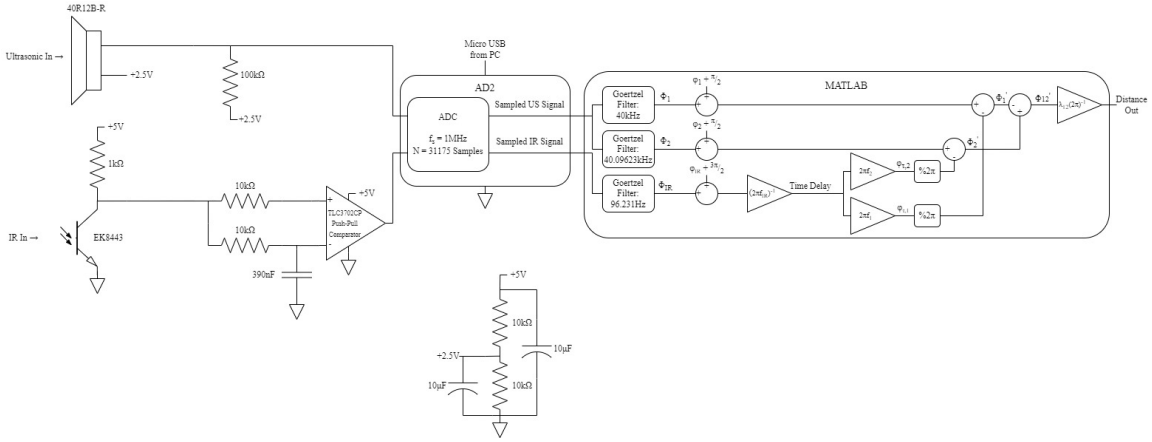

Figure 4.12: Schematic of Receiver Using AD2 and MATLAB

As in Section 3, the phase correction parameter, φ, is used to account for deterministic error associated with the channel and the air. Note that due to the π phase shift introduced by the IR receiver circuit as seen in Fig. 4.11, φ_{IR} has $\frac{3\pi}{2}$ added to it rather than $\frac{\pi}{2}$. To find φ experimentally, distance measurements are taken at known distances between the receiver and transmitter. For simplicity, φ_1 and φ_2 are the same as used in Section 3.3 and φ_{IR} is modified to produce the correct distance measurement. Since the measurement uses the relative phases of the signals, the individual phase measurements need not be correct, and correcting the IR signal's phase is equivalent to correct the two-tone signal's phase.

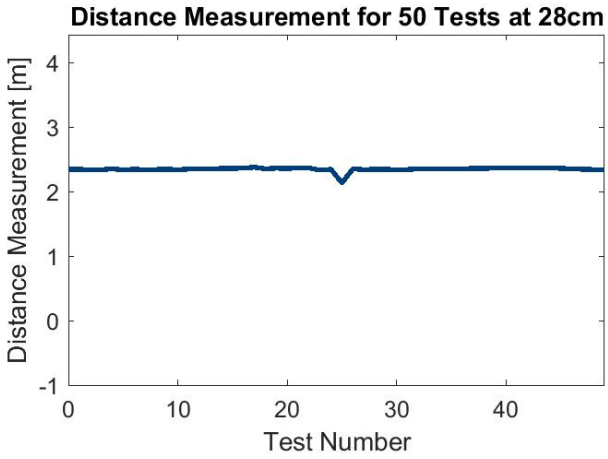

Figure 4.13: Measured Distances With the Receiver Fixed at 28cm Away from the Transmitter Without φ Correction

Fig. 4.13 shows the measured distance of 50 tests at a fixed receiver distance of 28cm without any φ_{IR} correction, where each test corresponds to a captured block of N = 31175 samples. Increasing φ_{IR} until the distance measurement is correct yields the following φ values.

Table 4.4: Phase Shifts, φ, for AD2 Receiver Implementation in Units of Radians

$\varphi_{a,IR}$	$\varphi_{a,1}$	$\varphi_{a,2}$
2.65	0.9838	0.9858

Using these phase corrections, the correct distance measurements can be obtained as in Fig. 4.14.

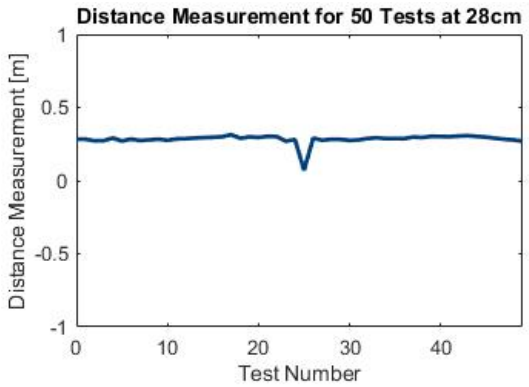

Figure 4.14: Measured Distances With the Receiver Fixed at 28cm Away from the Transmitter With φ Correction

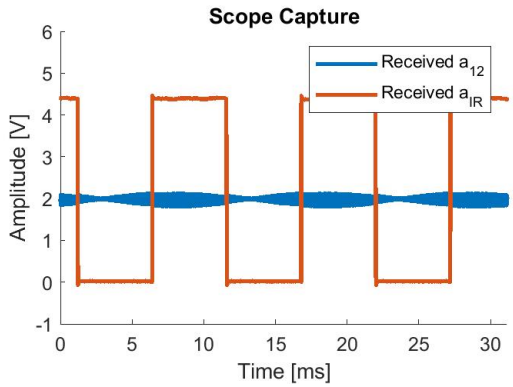

Figure 4.15: Scope Capture of Received Signals at 28cm Distance, Test 0

Fig. 4.15 shows a scope capture for test 0 at 28cm with N = 31175 samples collected at f_s = 1MHz. The

time shift τ shifts both signals equally and is subtracted out of the ultrasonic signal to extract a distance measurement as shown in Section 3.

The histogram of the distance measurements in Fig. 4.14 is shown in Fig. 4.16. The measurements have a fairly tight spread, with one outlier due to the spike in the data at around test 25.

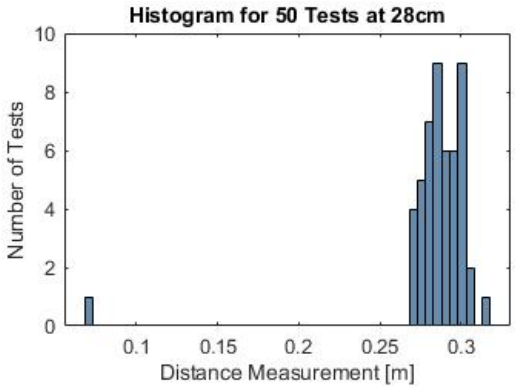

Figure 4.16: Histogram of 28cm Experiment

Fig. 4.17 shows the physical test setup for the 28cm experiment.

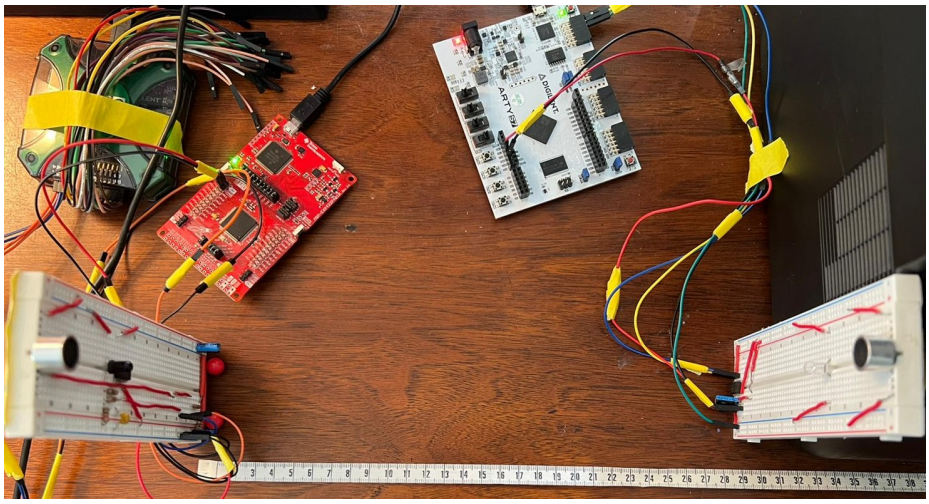

Figure 4.17: Photograph of 28cm Experiment

The implementation is then tested at a fixed distance of 22cm between the receiver and transmitter using the same φ correction. The reason these two distances are chosen for test data is because they are the lengths of the paper tubes created to isolate both channels (US and IR) from the environment during testing. However, these experiments are performed without the channel-isolating tubes for a more realistic representation of the system.

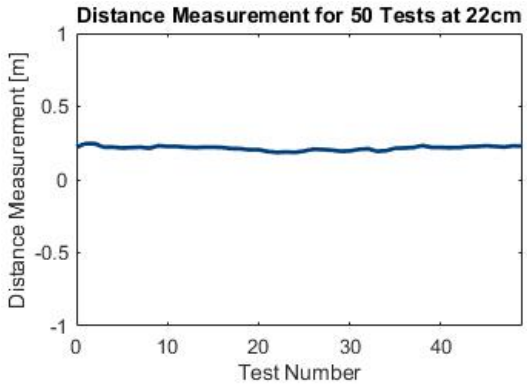

Figure 4.18: Measured Distances With the Receiver Fixed at 22cm Away from the Transmitter With φ Correction

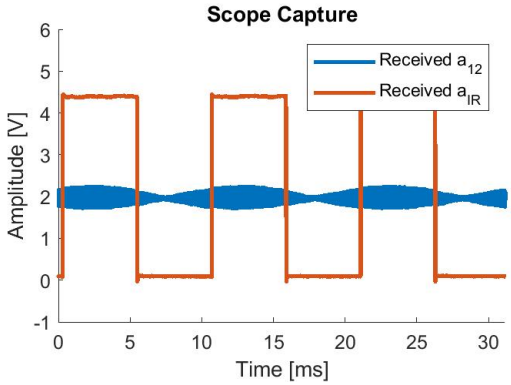

Figure 4.19: Scope Capture of Received Signals at 22cm Distance, Test 0

The histogram for the 22cm experiment in Fig. 4.20 shows an even tighter spread due to the lack of an outlier data point. In this scenario, the tighter the spread of the histogram the better, as it shows that the system is able to maintain a constant distance measurement when the receiver is kept at a constant distance from the receiver.

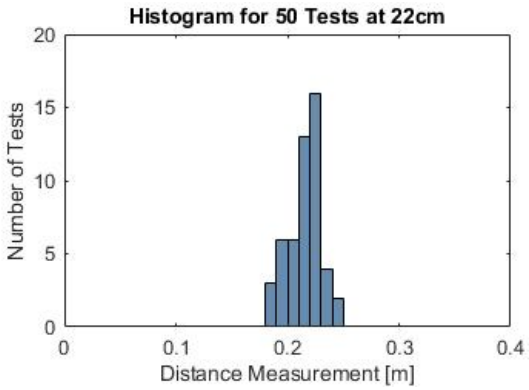

Figure 4.20: Histogram of 22cm Experiment

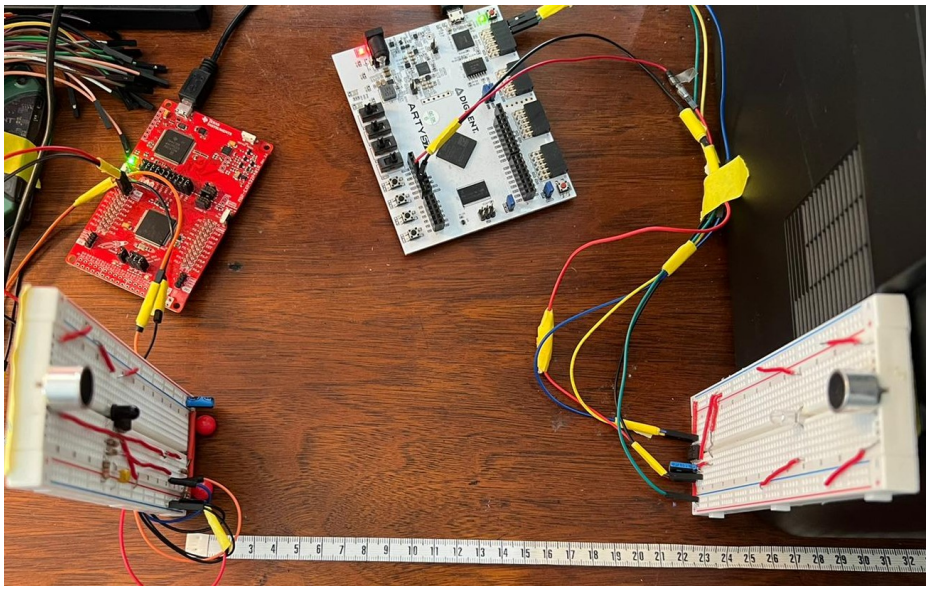

Figure 4.21: Photograph of 22cm Experiment

Both distance experiments yield solid results with only one small dip in the 28cm tests. The means and variances of the results of the two experiments are shown in Table 4.5.

Table 4.5: Means (μ) and Variances (σ^2) of 28cm and 22cm Experiments

Distance	μ	**Error**$_\mu$	σ^2
28cm	28.53cm	0.53cm	0.11cm
22cm	22.37cm	0.37cm	0.05cm

The slightly higher variance exhibited by the 28cm experiment is likely a result of the dip in distance measurements around test 25. The cause of this dip could be a bad capture from the ADC or a momentary disturbance in the operation of the circuit. The two fixed distances were measured using a tape measure and are thus subject to human error, therefore the best indication that the system is functioning properly is a low variance in the measurement data. This means that the synchronization is working properly, evidenced by the fact that τ is different (increasing) for each test, but the measurement stays nearly constant.

The scope captures in Fig. 4.15 and Fig. 4.19 show the attenuation of the ultrasonic signal as the distance increases. The amplitude of a_{12} at a distance of 22cm is about $0.8V_{PP}$ while the amplitude of a_{12} at 28cm is about $0.35V_{PP}$. This attenuation is not an issue at small distances, but for distances larger than about 32cm, the signal becomes small ans noisy enough to affect the accuracy of the measurements. Also at this distance, the IR signal begins to become unreliable likely due to the low-power, inexpensive diode and phototransistor. These effects are discussed further in Section 5.3.

Next, the receiver-transmitter link is tested by moving the receiver away from the transmitter starting at a distance of 22cm and ending at 32cm during data capture, again with the same φ correction.

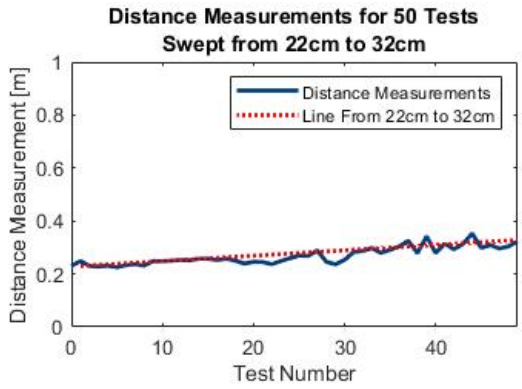

Figure 4.22: Measured Distances Obtained by Moving the Receiver a Distance of 22cm to 32cm Away from the Transmitter With φ Correction

Fig. 4.22 shows the result of this experiment as well as the theoretical distance measurements modeled by a line from 22cm to 32 cm. The experimental data matches this line fairly well, but the measurements do exhibit larger variance as the distance increases. This is likely due to noisy data resulting from moving the transmitter and its wires.

4.2.2 Effect of Non-Orthogonality

The preceding three experimental distance measurements are all taken immediately after programming the FPGA. This is important because it more closely resembles the simulations in Section 3 where the signals are generated at $t = \tau$ and τ ranges from 0 to 10ms. In the real-world, however, τ never stops increasing and becomes very large. This would have no effect if the signals were perfectly orthogonal, because they would always align to 0 phase every beat period of 10ms. However, due to the slight error between the beat frequency of the two-tone ultrasonic signal and the synchronization frequency of the IR signal seen in Table 4.3, the signals begin to drift for large τ and no longer align to zero phase at each beat period. This misalignment is cyclical with a frequency of $6.1736 * 10^{-5}$ Hz, the error between the beat frequency and the synchronization frequency. This means that approximately every 16198 seconds, or about 4.5 hours, the two signals align momentarily and then begin to drift apart again.

This error appears as a slope in the data when taken at a fixed distance over a large period of time. Fig. 4.23 shows this effect for 10000 tests at a fixed distance of 28cm.

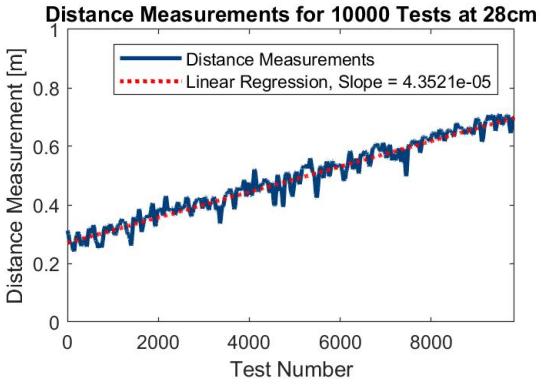

Figure 4.23: Measured Distances With the Receiver Fixed at 28cm Away from the Transmitter With φ Correction, Extended Capture Period

To model this error, linear regression is performed on the distance measurements to obtain the approximate slope, $4.3521 * 10^{-5}\ \frac{\text{m}}{\text{test}}$. By measuring the elapsed time of the data capture at approximately 2012 seconds, this slope can be converted to units of $\frac{\text{m}}{\text{s}}$.

$$\text{Slope} = 4.3521 * 10^{-5}\ \frac{\text{m}}{\text{test}} * \frac{10000\ \text{tests}}{2012\ \ \text{s}} = 2.1631 * 10^{-4}\ \frac{\text{m}}{\text{s}} \tag{4.8}$$

Finally, to measure the experimental error frequency, the slope is divided by the beat wavelength.

$$\text{Experimental Error Frequency} = 2.1631 * 10^{-4}\ \frac{\text{m}}{\text{s}} * \frac{1}{\lambda_{12}\ \text{m}} = 2.1631 * 10^{-4}\ \frac{\text{m}}{\text{s}} * \frac{1}{3.5643\ \text{m}} \tag{4.9}$$

$$\text{Experimental Error Frequency} = 6.0687 * 10^{-5}\ \text{Hz} \tag{4.10}$$

This measured value of the frequency error between the beat frequency of the ultrasonic signal and the synchronization frequency of the IR signal is very close to the analytical error frequency of 6.1736*10^{-5}Hz found in Table 4.3. This suggests that the theorized error associated with signals that aren't perfectly orthogonal translates into almost exactly the same error in the hardware implementation.

4.2.3 MSP432-based Receiver

In this section, the system is implemented using the MSP432 microcontroller as illustrated in the schematic in Fig. 4.24.

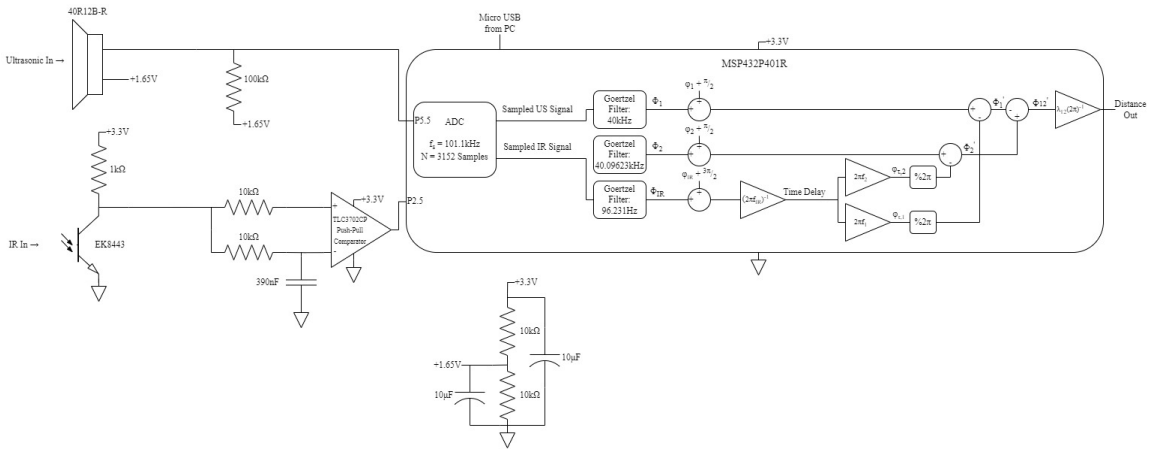

Figure 4.24: Schematic of Receiver Using MSP432P401R

The MSP432 implements the same receiver shown in the simulation block diagram in Fig. 3.17. The MSP432's onboard ADC is used to capture approximately 3 beat periods of the two-tone signal through pin 5.4, and the IR synchronization signal is captured by checking digital pin 2.5 for logic high or logic low upon capturing a sample in the ADC. This ensures the sampling rate is the same for both signals while not having to capture the IR signal using another ADC channel, which would reduce the performance of the system.

Using an ADC resolution of 10 bits, the sampling rate was found to be 101.1kHz by observing approximately 1011 samples in one period of a 100Hz signal generated by the AD2. Then, N must be calculated such that 3 beat periods are captured with the given sampling rate:

$$\mathrm{N} = \mathrm{int}(3 * \frac{\mathrm{f_s}}{\mathrm{f_{12}}}) = \mathrm{int}(3 * \frac{101.1\mathrm{kHz}}{96.231\mathrm{Hz}}) = 3152 \tag{4.11}$$

This gives the following number of periods of each of the signals:

$$\text{Number of Periods in IR Signal} = \frac{3152 * \mathrm{f_{IR}}}{\mathrm{f_s}} = \frac{3152 * 96.231\mathrm{Hz}}{101.1\mathrm{kHz}} = 3.0002 \text{ Periods} \tag{4.12}$$

$$\text{Number of Periods in } \mathrm{a_1} = \frac{3152 * 40\mathrm{kHz}}{101.1\mathrm{kHz}} = 1247.082 \text{ Periods} \tag{4.13}$$

$$\text{Number of Periods in } \mathrm{a_2} = \frac{3152 * 40.09623095\mathrm{kHz}}{101.1\mathrm{kHz}} = 1250.082 \text{ Periods} \tag{4.14}$$

$$\text{Number of Periods in } a_{12} = \frac{3152 * 96.23095\text{Hz}}{101.1\text{kHz}} = 3.0002 \text{ Periods} \tag{4.15}$$

The error in the number of periods calculated in the preceding equations appear as phase measurement error in the Goertzel Filters. This results in a higher variance in the distance measurements.

The distance measurements are taken at 22cm and observed by setting a breakpoint in the MSP432's code within Code Composer Studio (CCS), its Integrated Design Environment (IDE). The software then allows the user to print out the value of a variable, in this case distance, within the code. These distance measurements are then plotted using MATLAB in Fig. 4.25 for 20 tests.

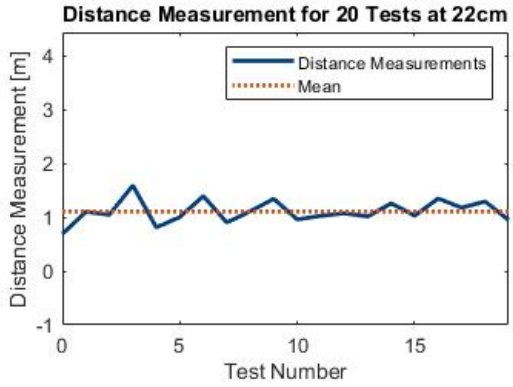

Figure 4.25: Measured Distances With the Receiver Fixed at 22cm Away from the Transmitter Using MSP432 Receiver

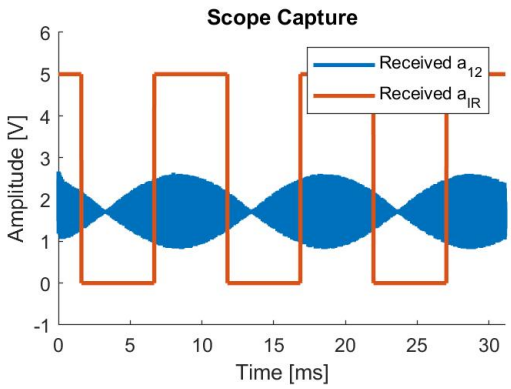

Figure 4.26: Plot of Received Signals at 22cm Distance Using MSP432 Receiver, Test 0

The IDE also exports values of arrays as CSV files, allowing the received signals for test 0 to be plotted using MATLAB in Fig. 4.26.

Table 4.6: Means (μ) and Variances (σ^2) of 22cm Experiment with MSP432 Receiver

Distance	μ	**Error**$_\mu$	σ^2
22cm	110.4cm	88.4cm	4.62cm

While the plot of test 0 and the distance measurements for the 22cm experiment looked promising, it was found that the experiment was not repeatable after a certain amount of time elapsed. This was not due to the imperfect orthogonality of the signals, however, because then the distance measurements would be observed to have the same approximate 6*10⁻5 Hz slope that was found in the AD2 receiver in Section 4.2.1. Instead, the distance measurements became seemingly random and uncorrelated with high variance after roughly 10

minutes of taking data. Upon inspection of the signals, the sampling rate of the MSP432's ADC had dropped to approximately 97.1kHz. This value is even closer to the Nyquist sampling rate for the fastest tone (approx. 80.2kHz for an approx. 40.1kHz tone), leading to aliasing of the sampled signal. This also caused large error in the Goertzel phase measurements, as the calculated constants were no longer based on the correct sampling rate and N captured less than beat periods of the signals.

One speculated reason for the variation in the sampling rate of the ADC is operating temperature of the device. The MSP432 microprocessor was observed to be very hot during operation, likely due to running at the highest available clock frequency and attempting to achieve the fastest performance. A higher operating temperature may also inject more noise into the ADC measurements. The sampling rate was also highly sensitive to the code, meaning adjusting resolution (by adjusting the number of decimal points in the constants) and adjusting where calculations are performed within the code affecting the sampling rate.

Because the measurements became unrepeatable over time, the correct values of φ were not able to be obtained, and therefore the $Error_{\mu}$ in Table 4.6 is large. However, the relatively low variance suggests a functioning system at least during the time the experiment was performed.

It is possible to calibrate the receiver to new sampling rates to once again obtain distance measurements with low variance. This is tedious, however, because changing N changes the amount of data that needs to be stored during runtime of the program. This effectively changes the load on the device, thus effecting the sampling rate. The found sampling rate of f_s = 101.1kHz and calculated N = 3152 samples was, at the time of the experiment, the perfect combination required to capture as close to 3 beat periods of the signal as possible. However, this required combination changes over time making the system unreliable.

5 Conclusion

The design proposed in this thesis introduces a low-cost solution to indoor localization using parts that can be easily obtained and implemented. The simulations showed that the design is viable and attains good position measurement accuracy, even in the face of time-discretization error due to transmitter clock frequency and sampling error due to finite sampling rate. By comparing the relative phase shifts of received signals, the system is robust against error and noise that all signals experience equally. With the multilateration method using averaging, the system becomes even less sensitive to common errors. In comparison to two of the most closely related works, this design was able to achieve similar accuracy to [4] in simulation and similar accuracy to [16] in hardware, while maintaining the advantage of using ultrasound over audible sound.

5.1 COVID-19 and Remote Work

Regretfully, the COVID-19 pandemic forced the work in this thesis to be done completely remotely, without access to Cal Poly facilities or in-person advising. The AD2, while capable for its purpose in this design, is not nearly as accurate or robust as the oscilloscopes, function generators, and other test equipment in the Cal Poly electrical engineering labs. The inability to debug hardware or observe data collaboratively was frustrating at times, and it lead to important errors being discovered late.

5.2 Reflection and Lessons Learned

Upon reflection of the work performed in this thesis, many lessons can be learned. Most importantly, if this work were to be performed again, more care would have been put into the generation of the signals in the system. Early in the design, the signals' frequencies were measured with low accuracy, and thus the imperfect orthogonality was discovered late and was unable to be accounted for in the final design. Had this work discovered the error associated with the signals' imperfect orthogonality earlier, more work could have been done to correct it. Also, in retrospect, more thought could have gone into the selection of the microcontrollers for both the transmitter and the receiver, seeing as the major drawbacks in this design are rooted in the microcontrollers. For example, an FPGA-based receiver might have been able to achieve a greater sampling rate and performance given less overhead compared to the C program used on the MSP432 in this design.

5.3 Future Work

5.3.1 Full IPS Implementation

The design proposed in this thesis has good potential for future expansion and improvement. The next step in the continuation of the design would be to implement all four transmitters in hardware as described. In addition, the receiver node could be optimized to only use one, capable microcontroller with a high sampling rate and data storage capability. This would effectively combine the two efforts made towards the receiver design in this thesis by wrapping a high-sampling-rate ADC and high-performance processor into one embedded device. The attempt at doing so in this thesis using the MSP432 microcontroller resulted in unfavorable results. It is highly recommended that a future implementation uses a receiver with a high and, more importantly, constant sampling rate such that the Goertzel Filters stay tuned to the correct frequencies and collect integer numbers of periods of the signals. Ideally, the Goertzel Filters could be implemented on an embedded device such that the device would not have to store N data points, just data points that represent the running parameters Q_0, Q_1, and Q_3. This would further lighten the system leading to possible improvements in speed, size, and/or power.

5.3.2 Better Tone Orthogonality

Another suggestion for future work on this topic is to create equally-spaced, orthogonal tones at the transmitter side, ensuring the system implements a proper OFDMA scheme. As mentioned in Section 4, the errors imposed by the non-ideal tone frequencies stemming from the resolution of the Arty S7 FPGA board are difficult to overcome. The simulation in Section 3.2 showed that the system works very well for equally-spaced tones. This can be achieved using a microcontroller board with a higher clock frequency or oscillator circuits tuned to the desired frequencies. As found in Section 4.2.2, the tones need only be spaced evenly ensuring they have equal beat frequencies, so long as the actual frequencies fall reasonably within the bandwidth of the ultrasonic channel and the exact beat frequency can be synthesized for synchronization.

5.3.3 Stronger Signals

As mentioned in Section 4.2.1, the received signals are very small in amplitude and not very reliable past a certain distance. Future work for this design would need to include more received signal conditioning to realize a system that can make distance measurements up to the full theoretical range of one beat wavelength

of the ultrasonic signal. This signal conditioning could come in the form of linear amplification and filtering of the received signals. The IR transducers were also observed to be highly sensitive to orientation, meaning the link had to line up a certain way for the signal to be properly transmitted. Devices with less directionality would function better for this design. For these reasons, an upgrade in IR devices is suggested so that the range of the system may increase.

5.3.4 Power Grid Synchronization

One interesting topic to explore in future work on this design is the possibility of synchronizing the tones of the system using the nominal frequency (60Hz for the United States) of the power grid that supplies the room with power. Electromagnetic energy at this frequency is emitted from power lines, outlets, and any metal connected to the grid. Therefore, by affixing an antenna to sense this 60Hz energy to the receiver and synchronizing the transmitters' tones to this same 60Hz energy, the system can be synchronized without the use of the IR link proposed in this design.

References

[1] Geotab Team, ”What is GPS?” Geotab, May 2020. [Online]. Available: https://www.geotab.com/blog/what-is-gps/. [Accessed: 2021].

[2] A. Witze, ”GPS Is Doing More Than You Thought,” Scientific American, October 2019. [Online]. Available: https://www.scientificamerican.com/article/gps-is-doing-more-than-you-thought/. [Accessed: 2021].

[3] C. Shaffer, ”GPS/GIS,” The International Encyclopedia of Primatology, April 2017. [Online]. Available: https://www.researchgate.net/publication/316658355_GPSGIS. [Accessed: 2021].

[4] K. Huang, “DISTANCE ESTIMATION USING OFDM SIGNALS FOR ULTRASONIC POSITIONING,” MSEE Thesis, Cal Poly Digital Commons, 2020.

[5] N. Rose, ”How ultrasonic local positioning system works,” Navigine, May 2021. [Online]. Available: https://navigine.com/blog/how-ultrasonic-indoor-positioning-works/. [Accessed: 2021].

[6] M. Gifford, ”Indoor Positioning with Ultrasonic/Ultrasound,” Navigine, October 2018. [Online]. Available: https://www.leverege.com/blogpost/ultrasonic-indoor-positioning. [Accessed: 2021].

[7] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, “The Cricket Location-Support System,” MIT Laboratory for Computer Science, 2004. [Online]. Available: http://nms.lcs.mit.edu/papers/cricket.pdf. [Accessed: 2021].

[8] “Sonitor Sense Technology Brief,” Sonitor. [Online]. Available: https://static1.squarespace.com/static/59cac734cf81e0d666427339/t/5f6b728fdcc756640a9bb36f/1600877203872/092320+Sonitor-Sonitor-Sense+Brief.pdf. [Accessed: 2021].

[9] J. Qi and G.-P. Liu, “A robust high-accuracy ultrasound indoor positioning system based on a wireless sensor network,” Sensors (Basel, Switzerland), 06-Nov-2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5713469/. [Accessed: 2021].

[10] W. Jiang and W. M. D. Wright, “Indoor Airborne Ultrasonic Wireless Communication Using OFDM Methods,” IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control (Volume: 64, Issue: 9 11-July-2017. [Online]. Available: https://ieeexplore.ieee.org/document/7973160. [Accessed: 2021].

[11] B. Ray, ”Wi-Fi Indoor Positioning Systems: The Good, The Bad & The Alternatives,” Link Labs, Feb. 2020. [Online]. Available: https://www.link-labs.com/blog/wifi-indoor-positioning-systems-pros-cons. [Accessed: 2021].

[12] “Ultrasound,” Wikipedia, 03-Aug-2021. [Online]. Available: https://en.wikipedia.org/wiki/Ultrasound. [Accessed: 2021].

[13] R. Nave, ”Sound Speed in Gases,” Georgia State University , 2019. [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/souspe3.html. [Accessed 2021].

[14] *Ultrasonic Sensor Set 40Khz Transmitter and Receiver*, Jameco Electronics. Accessed: 2021. [Online]. Available: https://www.jameco.com/z/40TR12B-R-Jameco-Valuepro-Ultrasonic-Sensor-Set-40Khz-Transmitter-and-Receiver_139492.html.

[15] ”Code Division Multiple Access (CDMA),” Network Encyclopedia. [Online]. Available: https://networkencyclopedia.com/code-division-multiple-access-cdma/. [Accessed: 2021]

[16] N. Luong, “Indoor Positioning Using Acoustic Pseudo-Noise Based Time Difference of Arrival,” MSEE Thesis, Cal Poly Digital Commons, 2020.

[17] L. Williams, “Fast Chirped Signals for a TDMA Ultrasonic Indoor Positioning System,” MSEE Thesis, Cal Poly Digital Commons, 2020.

[18] H. Dun, C. Tiberius, and G. Janssen, “Positioning based on OFDM signals through phase measurements,” Google, 2018. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:971a17cd-8755-4438-875d-a78a83ce3717/datastream/OBJ/download. [Accessed: 2021].

[19] “Orthogonal frequency-division multiplexing,” Wikipedia, 30-Sep-2021. [Online]. Available: https://en.wikipedia.org/wiki/Orthogonal_frequency-division_multiplexing. [Accessed: 2021].

[20] Keysight, Concepts of orthogonal frequency division multiplexing (OFDM) and 802.11 wlan. [Online]. Available: https://rfmw.em.keysight.com/wireless/helpfiles/89600B/WebHelp/Subsystems/wlan-ofdm/Content/ofdm_basicprinciplesoverview.html. [Accessed: 2021].

[21] ”How to calculate the perceived frequency of two sinusoidal waves added together?” Mathematics Stack-Exchange, 2012. [Online]. Available: https://math.stackexchange.com/questions/164369/how-to-calculate-the-perceived-frequency-of-two-sinusoidal-waves-added-together/164385. [Accessed: 2021].

[22] H. Yucel, R. Edizkan, T. Ozkir and A. Yazici, ”Development of indoor positioning system with ultrasonic and infrared signals,” 2012 International Symposium on Innovations in Intelligent Systems and Applications, 2012, pp. 1-4, doi: 10.1109/INISTA.2012.6246983.

[23] ”Why can’t I see very low frequencies in the Spectrum FFT window in LabChart?” ADInstruments. [Online]. Available: https://www.adinstruments.com/support/knowledge-base/why-cant-i-see-very-low-frequencies-spectrum-fft-window-labchart. [Accessed: 2021].

[24] Embedded Staff, “The Goertzel Algorithm,” Embedded, 28-Aug-2002. [Online]. Available: https://www.embedded.com/the-goertzel-algorithm/. [Accessed: 2021]

[25] T. Smyth, ”Nyquist Sampling Theorem,” UCSD, Oct. 2019. [Online]. Available: http://musicweb.ucsd.edu/~trsmyth/digitalAudio171/Nyquist_Sampling_Theorem.html. [Accessed: 2021].

[26] *Arty S7 Reference Manual*, Digilent. Accessed: 2021. [Online]. Available: https://digilent.com/reference/programmable-logic/arty-s7/reference-manual.

[27] Multiple Commenters, Gikfun 5mm 940nm LEDs Infrared Emitter and IR Receiver Diode for Arduino (Pack of 20pcs) EK8443. [Online]. Available: https://www.amazon.com/Gikfun-Infrared-Emitter-Receiver-Arduino/dp/B01HGIQ8NG. [Accessed: 2021].

[28] *TC4428A 1.5A Dual High-Speed Power MOSFET Drivers Datasheet*, Microchip. Accessed: 2021. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/20001423J.pdf.

[29] *Analog Discovery 2 Reference Manual*, Digilent. Accessed: 2021. [Online]. Available: https://digilent.com/reference/test-and-measurement/analog-discovery-2/reference-manual.

[30] T. Hula, “EE 449 Receiver Sub-System,” EE 449 Electronics Design Laboratory. California Polytechnic State University, San Luis Obispo, 2019.

[31] J. Bartolone, “EE 449 Transmitter Sub-System,” EE 449 Electronics Design Laboratory. California Polytechnic State University, San Luis Obispo, 2019.

[32] *LMx58-N Low-Power, Dual-Operational Amplifiers Datasheet*, Texas Instruments. Accessed: 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm358-n.pdf?ts=1635206535464&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM358-N.

[33] *MSP432P401R, MSP432P401M Mixed-Signal Microcontrollers Technical Reference Manual*, Texas Instruments, 2016. Accessed: 2021. [Online] Available: https://www.ti.com/lit/ds/slas826e/slas826e.pdf.

[34] *TLC3702 DUAL MICROPOWER LinCMOS™ VOLTAGE COMPARATORS Datasheet*, Texas Instruments. Accessed: 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/tlc3702.pdf?ts=1634608684967&ref_url=https%253A%252F%252Fwww.google.com%252F.

Appendix

A MATLAB Simulation Code

Listing A.1: Ideal Simulation Code

```
%% Clear
clear; close all; clc;
NumTests = 1;

%% Plot Stuff
Yellow = '#7B7A00';
Blue   = '#003F7B';
Green  = '#3E7B00';
Red    = '#7B0000';
Orange = '#FFA449';
Purple = '#CE8AFF';
MakePlots = 1;

%% Parameters
a1Frequency    = 40000;
a2Frequency    = 40100;
b1Frequency    = 39800;
b2Frequency    = 39900;
c1Frequency    = 39600;
c2Frequency    = 39700;
d1Frequency    = 39400;
d2Frequency    = 39500;
BeatFrequency  = 100;

SamplingRate   = 1E+6;
N              = 300000;
t              = (0:N-1)./SamplingRate;

for i = 1:NumTests
    %% Initialize Room and Object
    VSound          = 343;
    BeatWavelength  = VSound/BeatFrequency;
    MaxDistance     = sqrt(((BeatWavelength-0.01)^2)/2);
    RoomLength      = 0:0.0001:MaxDistance;
    RoomWidth       = 0:0.0001:MaxDistance;
```

```
36      ObjectPosition  = [MaxDistance*rand(1) MaxDistance*rand(1)];
37      aDistance       = sqrt(ObjectPosition(1)^2 + ObjectPosition(2)^2);
38      bDistance       = sqrt((MaxDistance-ObjectPosition(1))^2 + ObjectPosition(2)^2);
39      cDistance       = sqrt((MaxDistance-ObjectPosition(1))^2 +
    ↪  (MaxDistance-ObjectPosition(2))^2);
40      dDistance       = sqrt(ObjectPosition(1)^2 + (MaxDistance-ObjectPosition(2))^2);
41
42      if MakePlots == 1
43
44          figure(1); hold on; box on;
45          plot(ObjectPosition(1), ObjectPosition(2),'Marker', '.', 'MarkerSize', 20, 'Color',
    ↪  'black');
46          title('Room Setup: Four Transmitter Nodes, One Receiver Node');
47          xlabel('Distance [m]');
48          ylabel('Distance [m]');
49          xlim([0 RoomLength(end)]);
50          ylim([0 RoomLength(end)]);
51          text(ObjectPosition(1)-0.3, ObjectPosition(2)-0.15, ['Rx
    ↪  (',num2str(fix(10^4*ObjectPosition(1))/10^4),',
    ↪  ',num2str(fix(10^4*ObjectPosition(2))/10^4),')']);
52          text(0.05, 0.15, 'TXa');
53          text(RoomLength(end)-0.225, 0.15, 'TXb');
54          text(RoomLength(end)-0.225, RoomLength(end)-0.15, 'TXc');
55          text(0.05, RoomLength(end)-0.15, 'TXd');
56
57          plot(linspace(0, 0.3, 1000), linspace(0.3, 0.3, 1000), 'Linewidth', 2, 'Color',
    ↪  Blue);
58          plot(linspace(0.3, 0.3, 1000), linspace(0, 0.3, 1000), 'Linewidth', 2, 'Color',
    ↪  Blue);
59
60          plot(linspace(RoomLength(end)-0.3, RoomLength(end), 1000), linspace(0.3, 0.3,
    ↪  1000), 'Linewidth', 2, 'Color', Green);
61          plot(linspace(RoomLength(end)-0.3, RoomLength(end)-0.3, 1000), linspace(0, 0.3,
    ↪  1000), 'Linewidth', 2, 'Color', Green);
62
63          plot(linspace(RoomLength(end)-0.3, RoomLength(end), 1000),
    ↪  linspace(RoomLength(end)-0.3, RoomLength(end)-0.3, 1000), 'Linewidth', 2, 'Color',
    ↪  Red);
64          plot(linspace(RoomLength(end)-0.3, RoomLength(end)-0.3, 1000),
    ↪  linspace(RoomLength(end)-0.3, RoomLength(end), 1000), 'Linewidth', 2, 'Color', Red);
65
```

```
66          plot(linspace(0, 0.3, 1000), linspace(RoomLength(end)-0.3, RoomLength(end)-0.3,
    ↪  1000), 'Linewidth', 2, 'Color', Orange);
67          plot(linspace(0.3, 0.3, 1000), linspace(RoomLength(end)-0.3, RoomLength(end),
    ↪  1000), 'Linewidth', 2, 'Color', Orange);
68
69      end
70
71      %% Initialize Transmitters
72      TimeDelay   = (1/BeatFrequency)*rand(1);
73      [a12, aIR]  = TX(a1Frequency, a2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  aDistance);
74      [b12, ~]    = TX(b1Frequency, b2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  bDistance);
75      [c12, ~]    = TX(c1Frequency, c2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  cDistance);
76      [d12, ~]    = TX(d1Frequency, d2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  dDistance);
77
78      if MakePlots == 1
79
80          figure(2);
81          set(gcf, 'Position',  [100, 175, 750, 500]);
82
83          subplot(223);
84          plot(1000.*t, a12, 'LineWidth', 2, 'Color', Blue); hold on;
85          plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
86          title(['Transmitter a: f_1 = ', num2str(a1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(a2Frequency/1000), ' kHz']);
87          xlabel('t [msec]');
88          ylabel('Amplitude [V]');
89          legend({'a_{12}', 'a_{IR}'}, 'Location', 'northeast');
90          ylim([-2 8.6]);
91
92          subplot(224);
93          plot(1000.*t, b12, 'LineWidth', 2, 'Color', Green); hold on;
94          plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
95          title(['Transmitter b: f_1 = ', num2str(b1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(b2Frequency/1000), ' kHz']);
96          xlabel('t [msec]');
97          ylabel('Amplitude [V]');
98          legend({'b_{12}', 'a_{IR}'}, 'Location', 'northeast');
```

```
99          ylim([-2 8.6]);
100
101         subplot(222);
102         plot(1000.*t, c12, 'LineWidth', 2, 'Color', Red); hold on;
103         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
104         title(['Transmitter c: f_1 = ', num2str(c1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(c2Frequency/1000), ' kHz']);
105         xlabel('t [msec]');
106         ylabel('Amplitude [V]');
107         legend({'c_{12}', 'a_{IR}'}, 'Location', 'northeast');
108         ylim([-2 8.6]);
109
110         subplot(221);
111         plot(1000.*t, c12, 'LineWidth', 2, 'Color', Orange); hold on;
112         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
113         title(['Transmitter d: f_1 = ', num2str(d1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(d2Frequency/1000), ' kHz']);
114         xlabel('t [msec]');
115         ylabel('Amplitude [V]');
116         legend({'d_{12}', 'a_{IR}'}, 'Location', 'northeast');
117         ylim([-2 8.6]);
118
119         figure(3);
120         set(gcf, 'Position',  [100, 175, 750, 500]);
121
122         subplot(223);
123         plot(1000.*t, a12, 'LineWidth', 2, 'Color', Blue); hold on;
124         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
125         title(['Transmitter a: Zoom to 1^{st} Period']);
126         xlabel('t [msec]');
127         ylabel('Amplitude [V]');
128         legend({'a_{12}', 'a_{IR}'}, 'Location', 'northeast');
129         xlim([0 10*ceil(1/BeatFrequency)]);
130         ylim([-2 8.6]);
131
132         subplot(224);
133         plot(1000.*t, b12, 'LineWidth', 2, 'Color', Green); hold on;
134         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
135         title(['Transmitter b: Zoom to 1^{st} Period']);
136         xlabel('t [msec]');
137         ylabel('Amplitude [V]');
```

```
        legend({'b_{12}', 'a_{IR}'}, 'Location', 'northeast');
        xlim([0 10*ceil(1/BeatFrequency)]);
        ylim([-2 8.6]);

        subplot(222);
        plot(1000.*t, c12, 'LineWidth', 2, 'Color', Red); hold on;
        plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
        title(['Transmitter c: Zoom to 1^{st} Period']);
        xlabel('t [msec]');
        ylabel('Amplitude [V]');
        legend({'c_{12}', 'a_{IR}'}, 'Location', 'northeast');
        xlim([0 10*ceil(1/BeatFrequency)]);
        ylim([-2 8.6]);

        subplot(221);
        plot(1000.*t, c12, 'LineWidth', 2, 'Color', Orange); hold on;
        plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
        title(['Transmitter d: Zoom to 1^{st} Period']);
        xlabel('t [msec]');
        ylabel('Amplitude [V]');
        legend({'d_{12}', 'a_{IR}'}, 'Location', 'northeast');
        xlim([0 10*ceil(1/BeatFrequency)]);
        ylim([-2 8.6]);

    end

    %% Process Using Receiver
    aIRPhase  = wrapTo2Pi(GoertzelFilter(aIR, BeatFrequency, SamplingRate, N) + pi/2 +
↪   3.1415889e-04);
    a1Phase   = wrapTo2Pi(GoertzelFilter(a12, a1Frequency,   SamplingRate, N) + pi/2 +
↪   0.2519558);
    a2Phase   = wrapTo2Pi(GoertzelFilter(a12, a2Frequency,   SamplingRate, N) + pi/2 +
↪   0.2519558);
    b1Phase   = wrapTo2Pi(GoertzelFilter(b12, b1Frequency,   SamplingRate, N) + pi/2 +
↪   0.2506991);
    b2Phase   = wrapTo2Pi(GoertzelFilter(b12, b2Frequency,   SamplingRate, N) + pi/2 +
↪   0.2506991);
    c1Phase   = wrapTo2Pi(GoertzelFilter(c12, c1Frequency,   SamplingRate, N) + pi/2 +
↪   0.2494425);
    c2Phase   = wrapTo2Pi(GoertzelFilter(c12, c2Frequency,   SamplingRate, N) + pi/2 +
↪   0.2494425);
```

```
    d1Phase   = wrapTo2Pi(GoertzelFilter(d12, d1Frequency,   SamplingRate, N) + pi/2 +
↪   0.2481859);
    d2Phase   = wrapTo2Pi(GoertzelFilter(d12, d2Frequency,   SamplingRate, N) + pi/2 +
↪   0.2481859);

    if MakePlots == 1

        figure(4);
        set(gcf, 'Position',  [100, 175, (3/2)*750, 250]);

        subplot(131);
        stem(BeatFrequency, aIRPhase, 'LineWidth', 2, 'Color', Purple);
        title('Phase of a_{IR}');
        xlabel('Frequency [Hz]');
        ylabel('Phase [rad]');
        ylim([-1 2*pi+1]);

        subplot(132);
        stem([a1Frequency a2Frequency]./1000, [a1Phase a2Phase], 'LineWidth', 2, 'Color',
↪   Blue); hold on;
        stem([b1Frequency b2Frequency]./1000, [b1Phase b2Phase], 'LineWidth', 2, 'Color',
↪   Green);
        stem([c1Frequency c2Frequency]./1000, [c1Phase c2Phase], 'LineWidth', 2, 'Color',
↪   Red);
        stem([d1Frequency d2Frequency]./1000, [d1Phase d2Phase], 'LineWidth', 2, 'Color',
↪   Orange);
        title('Phase of All 8 Tones');
        xlabel('Frequency [Hz]');
        ylabel('Phase [rad]');
        legend({'a_{1,2}', 'b_{1,2}', 'c_{1,2}', 'd_{1,2}'}, 'Location', 'northeast');
        xlim([38.6 40.9]);
        ylim([-1 2*pi+1]);
    end

    %% Synchronize Tones
    [a1PhaseCorrected, a2PhaseCorrected, aPhaseSynchronized] = Synchronize(a1Phase,
↪   a2Phase, aIRPhase, a1Frequency, a2Frequency, BeatFrequency, SamplingRate);
    [b1PhaseCorrected, b2PhaseCorrected, bPhaseSynchronized] = Synchronize(b1Phase,
↪   b2Phase, aIRPhase, b1Frequency, b2Frequency, BeatFrequency, SamplingRate);
    [c1PhaseCorrected, c2PhaseCorrected, cPhaseSynchronized] = Synchronize(c1Phase,
↪   c2Phase, aIRPhase, c1Frequency, c2Frequency, BeatFrequency, SamplingRate);
```

```
204     [d1PhaseCorrected, d2PhaseCorrected, dPhaseSynchronized] = Synchronize(d1Phase,
    ↪ d2Phase, aIRPhase, d1Frequency, d2Frequency, BeatFrequency, SamplingRate);
205
206     if MakePlots == 1
207         figure(4);
208         subplot(133);
209         stem([a1Frequency a2Frequency]./1000, [a1PhaseCorrected a2PhaseCorrected],
    ↪ 'LineWidth', 2, 'Color', Blue); hold on;
210         stem([b1Frequency b2Frequency]./1000, [b1PhaseCorrected b2PhaseCorrected],
    ↪ 'LineWidth', 2, 'Color', Green);
211         stem([c1Frequency c2Frequency]./1000, [c1PhaseCorrected c2PhaseCorrected],
    ↪ 'LineWidth', 2, 'Color', Red);
212         stem([d1Frequency d2Frequency]./1000, [d1PhaseCorrected d2PhaseCorrected],
    ↪ 'LineWidth', 2, 'Color', Orange);
213         title('Synchronized Phase of All 8 Tones');
214         xlabel('Frequency [Hz]');
215         ylabel('Phase [rad]');
216         legend({'a_{1,2}', 'b_{1,2}', 'c_{1,2}', 'd_{1,2}'}, 'Location', 'northeast');
217         xlim([38.6 40.9]);
218         ylim([-1 2*pi+1]);
219     end
220
221     %% Measure Distances
222     aDistanceMeasure = PhaseToDistance(aPhaseSynchronized, BeatWavelength);
223     bDistanceMeasure = PhaseToDistance(bPhaseSynchronized, BeatWavelength);
224     cDistanceMeasure = PhaseToDistance(cPhaseSynchronized, BeatWavelength);
225     dDistanceMeasure = PhaseToDistance(dPhaseSynchronized, BeatWavelength);
226
227
228     figure(1);
229     Arc = 0:0.01:2*pi;
230     TXaCircle = plot(aDistanceMeasure*cos(Arc), aDistanceMeasure*sin(Arc), 'LineStyle',
    ↪ ':', 'LineWidth', 2, 'Color', Blue);
231     TXbCircle = plot(bDistanceMeasure*cos(Arc)+MaxDistance, bDistanceMeasure*sin(Arc),
    ↪ 'LineStyle', ':', 'LineWidth', 2, 'Color', Green);
232     TXcCircle = plot(cDistanceMeasure*cos(Arc)+MaxDistance,
    ↪ cDistanceMeasure*sin(Arc)+MaxDistance, 'LineStyle', ':', 'LineWidth', 2, 'Color',
    ↪ Red);
233     TXdCircle = plot(dDistanceMeasure*cos(Arc), dDistanceMeasure*sin(Arc)+MaxDistance,
    ↪ 'LineStyle', ':', 'LineWidth', 2, 'Color', Orange);
234
```

```
    [MeasuredPosition, abIntersection, bcIntersection, cdIntersection, daIntersection] =
↪   Multilaterate(TXaCircle, TXbCircle, TXcCircle, TXdCircle);

    Error = MeasuredPosition - ObjectPosition';

    if MakePlots ~= 1
        close all;
    end

    if MakePlots == 1
        figure(1);
        FillColor = 'green';
        if abs(Error(1)) +abs(Error(2)) > 0.5
            FillColor = Orange;
            if abs(Error(1))+abs(Error(2)) > 1
                FillColor = 'red';
            end
        end
        plot(MeasuredPosition(1), MeasuredPosition(2),'Marker', '.', 'MarkerSize', 20,
↪   'Color', FillColor);
        text(MeasuredPosition(1)-0.5, MeasuredPosition(2)+0.15, ['Measured Rx, Error: (',
↪   num2str(fix(10^4*Error(1))/10^4),', ', num2str(fix(10^4*Error(2))/10^4),')']);
        plot(abIntersection(1), abIntersection(2), 'Marker', 'x', 'Color', 'blue',
↪   'MarkerSize', 10, 'LineWidth', 2);
        plot(bcIntersection(1), bcIntersection(2), 'Marker', 'x', 'Color', 'blue',
↪   'MarkerSize', 10, 'LineWidth', 2);
        plot(cdIntersection(1), cdIntersection(2), 'Marker', 'x', 'Color', 'blue',
↪   'MarkerSize', 10, 'LineWidth', 2);
        plot(daIntersection(1), daIntersection(2), 'Marker', 'x', 'Color', 'blue',
↪   'MarkerSize', 10, 'LineWidth', 2);
    end

end
```

Listing A.2: System Simulation Code

```
%% Clear
clear; close all; clc;
NumTests = 1;

%% Plot Stuff
Yellow = '#7B7A00';
Blue   = '#003F7B';
Green  = '#3E7B00';
Purple = '#CE8AFF';
Orange = '#FFA449';
Red    = '#7B0000';
MakePlots = 1;

%% From FPGA Program
CLK     = 100000000;
a1Count = 1250;
a2Count = 1247;
b1Count = 1256;
b2Count = 1253;
c1Count = 1263;
c2Count = 1260;
d1Count = 1269;
d2Count = 1266;
IRCount = 519583;

a1Frequency    = 0.5*CLK/a1Count;
a2Frequency    = 0.5*CLK/a2Count;
b1Frequency    = 0.5*CLK/b1Count;
b2Frequency    = 0.5*CLK/b2Count;
c1Frequency    = 0.5*CLK/c1Count;
c2Frequency    = 0.5*CLK/c2Count;
d1Frequency    = 0.5*CLK/d1Count;
d2Frequency    = 0.5*CLK/d2Count;
BeatFrequency  = 0.5*CLK/IRCount;

%% Parameters
SamplingRate   = 101.1E+3;
N              = 3152;
```

```
t              = (0:N-1)./SamplingRate;

for i = 1:NumTests
    %% Initialize Room and Object
    VSound          = 343;
    BeatWavelength  = VSound/BeatFrequency;
    MaxDistance     = sqrt(((BeatWavelength-0.01)^2)/2);
    RoomLength      = 0:0.0001:MaxDistance;
    RoomWidth       = 0:0.0001:MaxDistance;
    ObjectPosition  = [MaxDistance*rand(1) MaxDistance*rand(1)];
    aDistance       = sqrt(ObjectPosition(1)^2 + ObjectPosition(2)^2);
    bDistance       = sqrt((MaxDistance-ObjectPosition(1))^2 + ObjectPosition(2)^2);
    cDistance       = sqrt((MaxDistance-ObjectPosition(1))^2 +
↪   (MaxDistance-ObjectPosition(2))^2);
    dDistance       = sqrt(ObjectPosition(1)^2 + (MaxDistance-ObjectPosition(2))^2);

    if MakePlots == 1

        figure(1); hold on; box on;
        plot(ObjectPosition(1), ObjectPosition(2),'Marker', '.', 'MarkerSize', 20, 'Color',
↪   'black');
        title('Room Setup: Four Transmitter Nodes, One Receiver Node');
        xlabel('Distance [m]');
        ylabel('Distance [m]');
        xlim([0 RoomLength(end)]);
        ylim([0 RoomLength(end)]);
        text(ObjectPosition(1)-0.3, ObjectPosition(2)-0.15, ['Rx
↪   (',num2str(fix(10^4*ObjectPosition(1))/10^4),',
↪   ',num2str(fix(10^4*ObjectPosition(2))/10^4),')']);
        text(0.05, 0.15, 'TXa');
        text(RoomLength(end)-0.225, 0.15, 'TXb');
        text(RoomLength(end)-0.225, RoomLength(end)-0.15, 'TXc');
        text(0.05, RoomLength(end)-0.15, 'TXd');

        plot(linspace(0, 0.3, 1000), linspace(0.3, 0.3, 1000), 'Linewidth', 2, 'Color',
↪   Blue);
        plot(linspace(0.3, 0.3, 1000), linspace(0, 0.3, 1000), 'Linewidth', 2, 'Color',
↪   Blue);

        plot(linspace(RoomLength(end)-0.3, RoomLength(end), 1000), linspace(0.3, 0.3,
↪   1000), 'Linewidth', 2, 'Color', Green);
```

```
73          plot(linspace(RoomLength(end)-0.3, RoomLength(end)-0.3, 1000), linspace(0, 0.3,
    ↪  1000), 'Linewidth', 2, 'Color', Green);
74
75          plot(linspace(RoomLength(end)-0.3, RoomLength(end), 1000),
    ↪  linspace(RoomLength(end)-0.3, RoomLength(end)-0.3, 1000), 'Linewidth', 2, 'Color',
    ↪  Red);
76          plot(linspace(RoomLength(end)-0.3, RoomLength(end)-0.3, 1000),
    ↪  linspace(RoomLength(end)-0.3, RoomLength(end), 1000), 'Linewidth', 2, 'Color', Red);
77
78          plot(linspace(0, 0.3, 1000), linspace(RoomLength(end)-0.3, RoomLength(end)-0.3,
    ↪  1000), 'Linewidth', 2, 'Color', Orange);
79          plot(linspace(0.3, 0.3, 1000), linspace(RoomLength(end)-0.3, RoomLength(end),
    ↪  1000), 'Linewidth', 2, 'Color', Orange);
80
81      end
82
83      %% Initialize Transmitters
84      TimeDelay   = (1/BeatFrequency)*rand(1);
85      [a12, aIR]  = TX(a1Frequency, a2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  aDistance);
86      [b12, ~]    = TX(b1Frequency, b2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  bDistance);
87      [c12, ~]    = TX(c1Frequency, c2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  cDistance);
88      [d12, ~]    = TX(d1Frequency, d2Frequency, BeatFrequency, SamplingRate, N, TimeDelay,
    ↪  dDistance);
89
90      if MakePlots == 1
91
92          figure(2);
93          set(gcf, 'Position',  [100, 175, 750, 500]);
94
95          subplot(221);
96          plot(1000.*t, a12, 'LineWidth', 2, 'Color', Blue); hold on;
97          plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
98          title(['Transmitter a: f_1 = ', num2str(a1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(a2Frequency/1000), ' kHz']);
99          xlabel('t [msec]');
100         ylabel('Amplitude [V]');
101         legend({'a_{12}', 'a_{IR}'}, 'Location', 'northeast');
102         ylim([-2 8.6]);
```

```
103
104         subplot(222);
105         plot(1000.*t, b12, 'LineWidth', 2, 'Color', Green); hold on;
106         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
107         title(['Transmitter b: f_1 = ', num2str(b1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(b2Frequency/1000), ' kHz']);
108         xlabel('t [msec]');
109         ylabel('Amplitude [V]');
110         legend({'b_{12}', 'a_{IR}'}, 'Location', 'northeast');
111         ylim([-2 8.6]);
112
113         subplot(223);
114         plot(1000.*t, c12, 'LineWidth', 2, 'Color', Red); hold on;
115         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
116         title(['Transmitter c: f_1 = ', num2str(c1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(c2Frequency/1000), ' kHz']);
117         xlabel('t [msec]');
118         ylabel('Amplitude [V]');
119         legend({'c_{12}', 'a_{IR}'}, 'Location', 'northeast');
120         ylim([-2 8.6]);
121
122         subplot(224);
123         plot(1000.*t, c12, 'LineWidth', 2, 'Color', Orange); hold on;
124         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
125         title(['Transmitter d: f_1 = ', num2str(d1Frequency/1000), ' kHz, f_2 = ',
    ↪  num2str(d2Frequency/1000), ' kHz']);
126         xlabel('t [msec]');
127         ylabel('Amplitude [V]');
128         legend({'d_{12}', 'a_{IR}'}, 'Location', 'northeast');
129         ylim([-2 8.6]);
130
131         figure(3);
132         set(gcf, 'Position',  [100, 175, 750, 500]);
133
134         subplot(221);
135         plot(1000.*t, a12, 'LineWidth', 2, 'Color', Blue); hold on;
136         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
137         title(['Transmitter a: Zoom to 1^{st} Period']);
138         xlabel('t [msec]');
139         ylabel('Amplitude [V]');
140         legend({'a_{12}', 'a_{IR}'}, 'Location', 'northeast');
```

```
141         xlim([0 10*ceil(1/BeatFrequency)]);
142         ylim([-2 8.6]);
143 
144         subplot(222);
145         plot(1000.*t, b12, 'LineWidth', 2, 'Color', Green); hold on;
146         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
147         title(['Transmitter b: Zoom to 1^{st} Period']);
148         xlabel('t [msec]');
149         ylabel('Amplitude [V]');
150         legend({'b_{12}', 'a_{IR}'}, 'Location', 'northeast');
151         xlim([0 10*ceil(1/BeatFrequency)]);
152         ylim([-2 8.6]);
153 
154         subplot(223);
155         plot(1000.*t, c12, 'LineWidth', 2, 'Color', Red); hold on;
156         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
157         title(['Transmitter c: Zoom to 1^{st} Period']);
158         xlabel('t [msec]');
159         ylabel('Amplitude [V]');
160         legend({'c_{12}', 'a_{IR}'}, 'Location', 'northeast');
161         xlim([0 10*ceil(1/BeatFrequency)]);
162         ylim([-2 8.6]);
163 
164         subplot(224);
165         plot(1000.*t, c12, 'LineWidth', 2, 'Color', Orange); hold on;
166         plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Purple); hold off;
167         title(['Transmitter d: Zoom to 1^{st} Period']);
168         xlabel('t [msec]');
169         ylabel('Amplitude [V]');
170         legend({'d_{12}', 'a_{IR}'}, 'Location', 'northeast');
171         xlim([0 10*ceil(1/BeatFrequency)]);
172         ylim([-2 8.6]);
173 
174     end
175 
176     %% Process Using Receiver
177     aIRPhase  = wrapTo2Pi(GoertzelFilter(aIR, BeatFrequency, SamplingRate, N) + pi/2 +
    ↪  0.0014946);
178     a1Phase   = wrapTo2Pi(GoertzelFilter(a12, a1Frequency,   SamplingRate, N) + pi/2 +
    ↪  0.9838190);
```

```
    a2Phase   = wrapTo2Pi(GoertzelFilter(a12, a2Frequency,   SamplingRate, N) + pi/2 + 0.9858353);
    b1Phase   = wrapTo2Pi(GoertzelFilter(b12, b1Frequency,   SamplingRate, N) + pi/2 + 0.8553438);
    b2Phase   = wrapTo2Pi(GoertzelFilter(b12, b2Frequency,   SamplingRate, N) + pi/2 + 0.9578522);
    c1Phase   = wrapTo2Pi(GoertzelFilter(c12, c1Frequency,   SamplingRate, N) + pi/2 + 0.4858836);
    c2Phase   = wrapTo2Pi(GoertzelFilter(c12, c2Frequency,   SamplingRate, N) + pi/2 + 0.7067454);
    d1Phase   = wrapTo2Pi(GoertzelFilter(d12, d1Frequency,   SamplingRate, N) + pi/2 - 0.0046242);
    d2Phase   = wrapTo2Pi(GoertzelFilter(d12, d2Frequency,   SamplingRate, N) + pi/2 + 0.3020730);

    if MakePlots == 1

        figure(4);
        set(gcf, 'Position',  [100, 175, (3/2)*750, 250]);

        subplot(131);
        stem(BeatFrequency, aIRPhase, 'LineWidth', 2, 'Color', Purple);
        title('Phase of a_{IR}');
        xlabel('Frequency [Hz]');
        ylabel('Phase [rad]');
        ylim([-1 2*pi+1]);

        subplot(132);
        stem([a1Frequency a2Frequency]./1000, [a1Phase a2Phase], 'LineWidth', 2, 'Color', Blue); hold on;
        stem([b1Frequency b2Frequency]./1000, [b1Phase b2Phase], 'LineWidth', 2, 'Color', Green);
        stem([c1Frequency c2Frequency]./1000, [c1Phase c2Phase], 'LineWidth', 2, 'Color', Red);
        stem([d1Frequency d2Frequency]./1000, [d1Phase d2Phase], 'LineWidth', 2, 'Color', Orange);
        title('Phase of All 8 Tones');
        xlabel('Frequency [Hz]');
        ylabel('Phase [rad]');
        legend({'a_{1,2}', 'b_{1,2}', 'c_{1,2}', 'd_{1,2}'}, 'Location', 'northeast');
        xlim([38.6 40.9]);
```

```
209         ylim([-1 2*pi+1]);
210     end
211
212     %% Synchronize Tones
213     [a1PhaseCorrected, a2PhaseCorrected, aPhaseSynchronized] = Synchronize(a1Phase,
    ↪  a2Phase, aIRPhase, a1Frequency, a2Frequency, BeatFrequency, SamplingRate);
214     [b1PhaseCorrected, b2PhaseCorrected, bPhaseSynchronized] = Synchronize(b1Phase,
    ↪  b2Phase, aIRPhase, b1Frequency, b2Frequency, BeatFrequency, SamplingRate);
215     [c1PhaseCorrected, c2PhaseCorrected, cPhaseSynchronized] = Synchronize(c1Phase,
    ↪  c2Phase, aIRPhase, c1Frequency, c2Frequency, BeatFrequency, SamplingRate);
216     [d1PhaseCorrected, d2PhaseCorrected, dPhaseSynchronized] = Synchronize(d1Phase,
    ↪  d2Phase, aIRPhase, d1Frequency, d2Frequency, BeatFrequency, SamplingRate);
217
218     if MakePlots == 1
219         figure(4);
220         subplot(133);
221         stem([a1Frequency a2Frequency]./1000, [a1PhaseCorrected a2PhaseCorrected],
    ↪  'LineWidth', 2, 'Color', Blue); hold on;
222         stem([b1Frequency b2Frequency]./1000, [b1PhaseCorrected b2PhaseCorrected],
    ↪  'LineWidth', 2, 'Color', Green);
223         stem([c1Frequency c2Frequency]./1000, [c1PhaseCorrected c2PhaseCorrected],
    ↪  'LineWidth', 2, 'Color', Red);
224         stem([d1Frequency d2Frequency]./1000, [d1PhaseCorrected d2PhaseCorrected],
    ↪  'LineWidth', 2, 'Color', Orange);
225         title('Synchronized Phase of All 8 Tones');
226         xlabel('Frequency [Hz]');
227         ylabel('Phase [rad]');
228         legend({'a_{1,2}', 'b_{1,2}', 'c_{1,2}', 'd_{1,2}'}, 'Location', 'northeast');
229         xlim([38.6 40.9]);
230         ylim([-1 2*pi+1]);
231     end
232
233
234     %% Measure Distances
235     aDistanceMeasure = PhaseToDistance(aPhaseSynchronized, BeatWavelength);
236     bDistanceMeasure = PhaseToDistance(bPhaseSynchronized, BeatWavelength);
237     cDistanceMeasure = PhaseToDistance(cPhaseSynchronized, BeatWavelength);
238     dDistanceMeasure = PhaseToDistance(dPhaseSynchronized, BeatWavelength);
239
240     figure(1);
241     Arc = 0:0.01:2*pi;
```

```
242     TXaCircle = plot(aDistanceMeasure*cos(Arc), aDistanceMeasure*sin(Arc), 'LineStyle',
    ↪ ':', 'LineWidth', 2, 'Color', Blue);
243     TXbCircle = plot(bDistanceMeasure*cos(Arc)+MaxDistance, bDistanceMeasure*sin(Arc),
    ↪ 'LineStyle', ':', 'LineWidth', 2, 'Color', Green);
244     TXcCircle = plot(cDistanceMeasure*cos(Arc)+MaxDistance,
    ↪ cDistanceMeasure*sin(Arc)+MaxDistance, 'LineStyle', ':', 'LineWidth', 2, 'Color',
    ↪ Red);
245     TXdCircle = plot(dDistanceMeasure*cos(Arc), dDistanceMeasure*sin(Arc)+MaxDistance,
    ↪ 'LineStyle', ':', 'LineWidth', 2, 'Color', Orange);
246
247     [MeasuredPosition, abIntersection, bcIntersection, cdIntersection, daIntersection] =
    ↪ Multilaterate(TXaCircle, TXbCircle, TXcCircle, TXdCircle);
248
249     Error = MeasuredPosition - ObjectPosition';
250
251     if MakePlots ~= 1
252         close all;
253     end
254
255     if MakePlots == 1
256         figure(1);
257         FillColor = 'green';
258         if abs(Error(1)) +abs(Error(2)) > 0.5
259             FillColor = Orange;
260             if abs(Error(1))+abs(Error(2)) > 1
261                 FillColor = 'red';
262             end
263         end
264         plot(MeasuredPosition(1), MeasuredPosition(2),'Marker', '.', 'MarkerSize', 20,
    ↪ 'Color', FillColor);
265         text(MeasuredPosition(1)-0.5, MeasuredPosition(2)+0.15, ['Measured Rx, Error: (',
    ↪ num2str(fix(10^4*Error(1))/10^4),', ', num2str(fix(10^4*Error(2))/10^4),')']);
266         plot(abIntersection(1), abIntersection(2), 'Marker', 'x', 'Color', 'blue',
    ↪ 'MarkerSize', 10, 'LineWidth', 2);
267         plot(bcIntersection(1), bcIntersection(2), 'Marker', 'x', 'Color', 'blue',
    ↪ 'MarkerSize', 10, 'LineWidth', 2);
268         plot(cdIntersection(1), cdIntersection(2), 'Marker', 'x', 'Color', 'blue',
    ↪ 'MarkerSize', 10, 'LineWidth', 2);
269         plot(daIntersection(1), daIntersection(2), 'Marker', 'x', 'Color', 'blue',
    ↪ 'MarkerSize', 10, 'LineWidth', 2);
270     end
```

```
271
272 end
273
274
275
276
277
```

Listing A.3: Goertzel Filter Function Code

```
function Phase = GoertzelFilter(Signal, Frequency, SamplingRate, N)

    K      = floor(N * Frequency / SamplingRate);
    W      = 2 * pi * K / N;
    Coeff  = 2 * cos(W);
    Q1     = 0;
    Q2     = 0;

    for Index = 1:N
        Q0  = Coeff * Q1 - Q2 + Signal(Index);
        Q2  = Q1;
        Q1  = Q0;
    end

    Real   = Q1 - Q2 * cos(W);
    Imag   = Q2 * sin(W);
    Phase  = wrapTo2Pi(atan2(Imag , Real));

end

```

Listing A.4: Synchronize Function Code

```
1  function [Tone1PhaseCorrected, Tone2PhaseCorrected, PhaseCorrected] =
   ↪ Synchronize(Tone1Phase, Tone2Phase, IRPhase, Tone1Frequency, Tone2Frequency,
   ↪ IRFrequency, SamplingRate)
2
3      MeasuredTimeOffset   = IRPhase/(2*pi*IRFrequency);
4      Tone1PhaseCorrected  = wrapTo2Pi(Tone1Phase -
   ↪ mod(2*pi*MeasuredTimeOffset*Tone1Frequency, 2*pi));
5      Tone2PhaseCorrected  = wrapTo2Pi(Tone2Phase -
   ↪ mod(2*pi*MeasuredTimeOffset*Tone2Frequency, 2*pi));
6      PhaseCorrected       = fix(SamplingRate/10 * wrapTo2Pi(Tone2PhaseCorrected -
   ↪ Tone1PhaseCorrected)) / (SamplingRate/10);
7
8  end
```

Listing A.5: TX Function Code

```
function [Sig12, SigIR] = TX(Tone1Frequency, Tone2Frequency, BeatFrequency, SamplingRate,
↪ N, TimeDelay, Distance)

    DCOffset     = 1.65;
    A            = 1.65;
    t            = (0:N-1) ./ SamplingRate;
    Lambda1      = 343 / Tone1Frequency;
    Lambda2      = 343 / Tone2Frequency;
    Tone1        = DCOffset + A * sin(2*pi*Tone1Frequency.*(t+TimeDelay) +
↪ 2*pi*Distance/Lambda1);
    Tone2        = DCOffset + A * sin(2*pi*Tone2Frequency.*(t+TimeDelay) +
↪ 2*pi*Distance/Lambda2);
    Sig12        = Tone1 + Tone2;
    SigIR        = 2*DCOffset + 2*A * square(2*pi*BeatFrequency.*(t+TimeDelay));

end
```

Listing A.6: Phase → Distance Function Code

```
function Distance = PhaseToDistance(Phase, BeatWavelength)

    Distance = BeatWavelength*Phase/(2*pi);

end
```

Listing A.7: Multilaterate Function Code

```
function [MeasuredPosition, abIntersection, bcIntersection, cdIntersection,
↪  daIntersection]    = Multilaterate(TXaCircle, TXbCircle, TXcCircle, TXdCircle)

    aQuadrant = 1:ceil(length(TXaCircle.XData)/4);
    bQuadrant = ceil(length(TXaCircle.XData)/4):ceil(length(TXaCircle.XData)/2);
    cQuadrant = ceil(length(TXaCircle.XData)/2):ceil(3*length(TXaCircle.XData)/4);
    dQuadrant = ceil(3*length(TXaCircle.XData)/4):ceil(length(TXaCircle.XData));


    TXaCircle = fix(10^5.*[TXaCircle.XData(aQuadrant); TXaCircle.YData(aQuadrant)])./10^5;
    TXbCircle = fix(10^5.*[TXbCircle.XData(bQuadrant); TXbCircle.YData(bQuadrant)])./10^5;
    TXcCircle = fix(10^5.*[TXcCircle.XData(cQuadrant); TXcCircle.YData(cQuadrant)])./10^5;
    TXdCircle = fix(10^5.*[TXdCircle.XData(dQuadrant); TXdCircle.YData(dQuadrant)])./10^5;

    abVectors = 10.*ones(length(TXaCircle), length(TXaCircle));
    bcVectors = 10.*ones(length(TXbCircle), length(TXbCircle));
    cdVectors = 10.*ones(length(TXcCircle), length(TXcCircle));
    daVectors = 10.*ones(length(TXdCircle), length(TXdCircle));

    for aIndex = 1:length(TXaCircle)
        for bIndex = 1:length(TXbCircle)

            abVectors(aIndex, bIndex) = sqrt((TXaCircle(1, aIndex) - TXbCircle(1,
↪  bIndex))^2 + (TXaCircle(2, aIndex) - TXbCircle(2, bIndex))^2);
            bcVectors(aIndex, bIndex) = sqrt((TXbCircle(1, aIndex) - TXcCircle(1,
↪  bIndex))^2 + (TXbCircle(2, aIndex) - TXcCircle(2, bIndex))^2);
            cdVectors(aIndex, bIndex) = sqrt((TXcCircle(1, aIndex) - TXdCircle(1,
↪  bIndex))^2 + (TXcCircle(2, aIndex) - TXdCircle(2, bIndex))^2);
            daVectors(aIndex, bIndex) = sqrt((TXdCircle(1, aIndex) - TXaCircle(1,
↪  bIndex))^2 + (TXdCircle(2, aIndex) - TXaCircle(2, bIndex))^2);

            if abVectors(aIndex, bIndex) == min(abVectors(:))
                abIntersection = TXaCircle(:, aIndex);
            end

            if bcVectors(aIndex, bIndex) == min(bcVectors(:))
                bcIntersection = TXbCircle(:, aIndex);
            end
```

```
34
35              if cdVectors(aIndex, bIndex) == min(cdVectors(:))
36                  cdIntersection = TXcCircle(:, aIndex);
37              end
38
39              if daVectors(aIndex, bIndex) == min(daVectors(:))
40                  daIntersection = TXdCircle(:, aIndex);
41              end
42
43          end
44
45      end
46
47      MeasuredPosition = (abIntersection + bcIntersection + cdIntersection +
    ↪  daIntersection)./4;
48
49  end
```

B Transmitter Code

Listing B.1: Transmitter FPGA Verilog Code

```
`timescale 1ns / 1ps

module sq_wave_gen(
    input CLK100MHZ,
    input reset_n,
    output sq_wave40k,
    output sq_wave40_1k,
    output ir
    );

    parameter count_40k    = 1249;
    parameter count_40_1k  = 1246;
    parameter count_ir     = 519582;

    reg [31:0] counter_40k;
    reg [31:0] counter_40_1k;
    reg [31:0] counter_ir;

    reg sq_wave40k_reg;
    reg sq_wave40_1k_reg;
    reg ir_reg;

    always @ (posedge CLK100MHZ, posedge reset_n)
    if (reset_n)
        begin
            sq_wave40k_reg    <= 0;
            sq_wave40_1k_reg  <= 0;
            ir_reg            <= 0;
            counter_40k       <= 0;
            counter_40_1k     <= 0;
            counter_ir        <= 0;
        end
        else
            begin
                if (counter_40_1k == count_40_1k)
                    begin
                        sq_wave40_1k_reg <= !sq_wave40_1k_reg;
                        counter_40_1k    <= 0;
```

```
                    end
                else
                    begin
                        counter_40_1k    <= counter_40_1k + 1;
                    end

                if (counter_40k == count_40k)
                    begin
                        sq_wave40k_reg <= !sq_wave40k_reg;
                        counter_40k    <= 0;
                    end
                else
                    begin
                        counter_40k    <= counter_40k + 1;
                    end

                if (counter_ir == count_ir)
                    begin
                        ir_reg      <= !ir_reg;
                        counter_ir  <= 0;
                    end
                else
                    begin
                        counter_ir    <= counter_ir + 1;
                    end
            end

    assign sq_wave40k     = sq_wave40k_reg;
    assign sq_wave40_1k   = sq_wave40_1k_reg;
    assign ir             = ir_reg;

endmodule
```

C Receiver Code

Listing C.1: MATLAB System Code

```
%% Clear
clear; close all; clc;
NumTests = 50;

%% Plot Stuff
Blue   = '#003F7B';
Red    = '#7B0000';
MakePlots = 0;

Distances = zeros(1, NumTests);
for FileNum = 0:NumTests-1
    %% Initialize Signals
    CLK      = 100000000;
    a1Count  = 1250;
    a2Count  = 1247;
    aIRCount = 519583;

    a1Frequency = 0.5*CLK/a1Count;
    a2Frequency = 0.5*CLK/a2Count;

    SamplingRate    = 1E+6;
    BeatFrequency   = 0.5*CLK/aIRCount;
    N               = 31175;
    t               = (0:N-1)./SamplingRate;
    VSound          = 343;
    BeatWavelength  = VSound/BeatFrequency;

    a12 = readmatrix("ScopeData/US"+num2str(FileNum)+".csv");
    aIR = readmatrix("ScopeData/IR"+num2str(FileNum)+".csv");

    %% Process Using Receiver
    aIRPhase  = wrapTo2Pi(GoertzelFilter(aIR, BeatFrequency, SamplingRate, N) + 3*pi/2 +
    ↪ 2.65);
    a1Phase   = wrapTo2Pi(GoertzelFilter(a12, a1Frequency,   SamplingRate, N) + pi/2   +
    ↪ 0.9838190);
    a2Phase   = wrapTo2Pi(GoertzelFilter(a12, a2Frequency,   SamplingRate, N) + pi/2   +
    ↪ 0.9858353);

```

```
    %% Synchronize Tones
    [a1PhaseCorrected, a2PhaseCorrected, aPhaseSynchronized] = Synchronize(a1Phase,
↪   a2Phase, aIRPhase, a1Frequency, a2Frequency, BeatFrequency, SamplingRate);

    %% Measure Distances
    Distances(FileNum+1) = PhaseToDistance(aPhaseSynchronized, BeatWavelength);

    if MakePlots == 1

        figure();
        set(gcf, 'Position',  [100, 175, 750, 250]);

        subplot(121);
        plot(1000.*t, a12, 'LineWidth', 2, 'Color', Blue); hold on;
        plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Red); hold off;
        title(['Transmitter a: f_1 = ', num2str(a1Frequency/1000), ' kHz, f_2 = ',
↪   num2str(a2Frequency/1000), ' kHz']);
        xlabel('t [msec]');
        ylabel('Amplitude [V]');
        legend({'a_{12}', 'a_{IR}'}, 'Location', 'northeast');
        ylim([-2 8.6]);

        subplot(122);
        plot(1000.*t, a12, 'LineWidth', 2, 'Color', Blue); hold on;
        plot(1000.*t, aIR, 'LineWidth', 2, 'Color', Red); hold off;
        title(['Transmitter a: Zoom to 1^{st} Period']);
        xlabel('t [msec]');
        ylabel('Amplitude [V]');
        legend({'a_{12}', 'a_{IR}'}, 'Location', 'northeast');
        xlim([0 10*ceil(1/BeatFrequency)]);
        ylim([-2 8.6]);

        figure();
        set(gcf, 'Position',  [100, 175, 750, 250]);

        subplot(121);
        stem(BeatFrequency, aIRPhase, 'LineWidth', 2, 'Color', Red);
        title('Phase of a_{IR}');
        xlabel('Frequency [Hz]');
        ylabel('Phase [rad]');
        ylim([-1 2*pi+1]);
```

```
75
76          subplot(122);
77          stem([a1Frequency a2Frequency]./1000, [a1Phase a2Phase], 'LineWidth', 2, 'Color',
    ↪  Blue);
78          title('Phase of All 8 Tones');
79          xlabel('Frequency [Hz]');
80          ylabel('Phase [rad]');
81          legend({'a_{1,2}'}, 'Location', 'northeast');
82          xlim([38.6 40.9]);
83          ylim([-1 2*pi+1]);
84
85      end
86
87  end
88
89  %% Plot
90  figure;
91  plot(0:length(Distances)-1, Distances, 'LineWidth', 2, 'Color', Blue); hold on;
92  set(gcf, 'Position',  [100, 175, (1/2)*750, 250]);
93  title("Distance Measurement for "+num2str(NumTests)+" Tests at 28cm");
94  xlabel("Test Number");
95  ylabel("Distance Measurement [m]");
96  ylim([-1 3.43+1]);
97  xlim([0 NumTests-1]);
```

Listing C.2: MSP432 Receiver Code

```
#include "stdint.h"
#include "msp.h"
#include "math.h"
#include "clock.h"

#define FREQ FREQ_48_MHZ

#define a1Frequency   40000.0
#define a2Frequency   40096.23
#define aIRFrequency  96.231

#define pi            3.1416
#define SamplingRate  101100
#define N             3152

float A0results[N];
int   A1results[N];

static uint16_t Index = 0;

int main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;  // Halt Watchdog Timer

    // Setup 48 MHz Clock
    init_dco();
    set_dco(FREQ);

    // GPIO Setup
    P2->SEL0 &= ~BIT5;
    P2->SEL1 &= ~BIT5;
    P2->DIR  &= ~BIT5; // Configure P2.5 for input

    P5->SEL1 |= BIT5;
    P5->SEL0 |= BIT5;  // Configure P5.5 (A0) for ADC

    // Sampling time, S&H=4, ADC14 on
```

```
39      ADC14->CTL0 = ADC14_CTL0_SHT0_0 | ADC14_CTL0_SHP | ADC14_CTL0_ON |ADC14_CTL0_SSEL_4;
40      ADC14->CTL1 = ADC14_CTL1_RES_0;         // Use sampling timer, 10-bit conversion
41
42      ADC14->MCTL[0] |= ADC14_MCTLN_INCH_0;  // A0 ADC input select; Vref=AVCC
43      ADC14->IER0    |=  ADC14_IER0_IE0;     // Enable ADC conversion complete interrupt
44      ADC14->CTL0    |= ADC14_CTL0_ENC;      // Enable conversions
45
46      // Enable ADC interrupt in NVIC module
47      NVIC->ISER[0] = 1 << ((ADC14_IRQn) & 31);
48
49      __enable_irq();
50
51      int K1    = N * a1Frequency / SamplingRate;
52      float W1  = 2 * pi * K1 / N;
53
54      int K2    = N * a2Frequency / SamplingRate;
55      float W2  = 2 * pi * K2 / N;
56
57      int K3    = N * aIRFrequency / SamplingRate;
58      float W3  = 2 * pi * K3 / N;
59
60      while (1)
61      {
62          ADC14->CTL0 |= ADC14_CTL0_SC;
63          if (Index == N) {
64
65              float Q11 = 0.0;
66              float Q21 = 0.0;
67
68              float Q12 = 0.0;
69              float Q22 = 0.0;
70
71              float Q13 = 0.0;
72              float Q23 = 0.0;
73
74              int i = 0;
75              for (i = 0; i < N; i++){
76                  float Q01 = 2*cos(W1) * Q11 - Q21 + A0results[i];;
77                  Q21 = Q11;
78                  Q11 = Q01;
79                  float Q02 = 2*cos(W2) * Q12 - Q22 + A0results[i];;
```

```
80                  Q22 = Q12;
81                  Q12 = Q02;
82                  float Q03 = 2*cos(W3) * Q13 - Q23 + A1results[i];;
83                  Q23 = Q13;
84                  Q13 = Q03;
85              }
86
87              float Real1   = Q11 - Q21 * cos(W1);
88              float Imag1   = Q21 * sin(W1);
89              float Mag1    = Real1 * Real1 + Imag1 * Imag1;
90              float Phase1  = atan2(Imag1, Real1) + pi/2 + 0.984;
91              if (Phase1 < 0) {
92                  Phase1 += 2*pi;
93              }
94
95              float Real2   = Q12 - Q22 * cos(W2);
96              float Imag2   = Q22 * sin(W2);
97              float Mag2    = Real2 * Real2 + Imag2 * Imag2;
98              float Phase2  = atan2(Imag2, Real2) + pi/2  + 0.986;
99              if (Phase2 < 0) {
100                 Phase2 += 2*pi;
101             }
102
103             float Real3   = Q13 - Q23 * cos(W3);
104             float Imag3   = Q23 * sin(W3);
105             float Mag3    = Real3 * Real3 + Imag3 * Imag3;
106             float Phase3  = atan2(Imag3, Real3) + 3*pi/2 + 0.0015;
107             if (Phase3 < 0) {
108                 Phase3 += 2*pi;
109             }
110
111             float TimeDelay        = Phase3/(2*pi*aIRFrequency);
112             float Delay1           = 2*pi*TimeDelay*a1Frequency;
113             int   Div1             = (int)floor(Delay1/(2*pi));
114             float Remainder1       = Delay1 - Div1*2*pi;
115             float Phase1Corrected  = Phase1 - Remainder1;
116             if ( Phase1Corrected < 0) {
117                 Phase1Corrected += 2*pi;
118             }
119             float Delay2           = 2*pi*TimeDelay*a2Frequency;
120             int   Div2             = (int)floor(Delay2/(2*pi));
```

```
            float Remainder2       = Delay2 - Div2*2*pi;
            float Phase2Corrected  = Phase2 - Remainder2;
            if ( Phase2Corrected < 0) {
                Phase2Corrected += 2*pi;
            }

            float PhaseDifference  = Phase2Corrected - Phase1Corrected;
            if ( PhaseDifference < 0) {
                PhaseDifference += 2*pi;
            }

            float Distance         = (343/(aIRFrequency))*PhaseDifference/(2*pi);

            Index = 0;
        }
    }
}

// ADC14 interrupt service routine
void ADC14_IRQHandler(void) {

    if (Index != N) {
        A0results[Index]  = ADC14->MEM[0];   // Move A0 results, IFG is cleared
        A1results[Index]  = P2->IN & BIT5;   // Check P2.5 for IR Signal
        Index             = (Index + 1);     // Increment results Index
    }
    else {
        ADC14->CLRIFGR0 |= BIT0 | BIT1;
    }
}
```

Listing C.3: MSP432 Receiver Clock Initialization Code

```
#include "msp.h"
#include "clock.h"

// Initialize DCO and other clock values
void init_dco(void){
    CS->KEY = CS_KEY_VAL;          // Unlock key
    CS->CTL0 = 0;                  // Reset
    CS->CTL0 = CS_CTL0_DCORSEL_3;  // 12 MHz
    CS->CTL1 = CS_CTL1_SELA_2 |    // Set ACLK to REFCLK
               CS_CTL1_SELS_3 |    // Set SMCLK to DCO
               CS_CTL1_DIVS__16 |  // Divide SMCLK
               CS_CTL1_SELM_3;     // Set MCLK to DCO
    CS->KEY = LOCK_CS_KEY;         // Lock key
}

// Apply safe settings for 48MHz operation
static inline void safe_48(){
    // Transition to VCORE Level 1: AM0_LDO --> AM1_LDO
    while ((PCM->CTL1 & PCM_CTL1_PMR_BUSY));
    PCM->CTL0 = PCM_CTL0_KEY_VAL | PCM_CTL0_AMR_1;
    while ((PCM->CTL1 & PCM_CTL1_PMR_BUSY));

    // Configure Flash wait-state to 1 for both banks 0 & 1
    FLCTL->BANK0_RDCTL = (FLCTL->BANK0_RDCTL &
     ~(FLCTL_BANK0_RDCTL_WAIT_MASK)) | FLCTL_BANK0_RDCTL_WAIT_1;
    FLCTL->BANK1_RDCTL = (FLCTL->BANK0_RDCTL &
     ~(FLCTL_BANK1_RDCTL_WAIT_MASK)) | FLCTL_BANK1_RDCTL_WAIT_1;
}

// Set DCO to provide frequency
void set_dco(unsigned int freq){
    CS->KEY = CS_KEY_VAL;  // Unlock key
    CS->CTL0 = 0;          // Reset
    switch(freq){
    case(FREQ_1_5_MHZ):
        CS->CTL0 = CS_CTL0_DCORSEL_0;
        break;
    case(FREQ_3_MHZ):
```

```
39          CS->CTL0 = CS_CTL0_DCORSEL_1;
40          break;
41      case(FREQ_6_MHZ):
42          CS->CTL0 = CS_CTL0_DCORSEL_2;
43          break;
44      case(FREQ_12_MHZ):
45          CS->CTL0 = CS_CTL0_DCORSEL_3;
46          break;
47      case(FREQ_24_MHZ):
48          CS->CTL0 = CS_CTL0_DCORSEL_4;
49          break;
50      case(FREQ_48_MHZ):
51          safe_48();  // Apply settings for safe 48MHz operation
52          CS->CTL0 = CS_CTL0_DCORSEL_5;
53          break;
54      default:
55          // Default to 1.5MHz
56          CS->CTL0 = CS_CTL0_DCORSEL_0;
57      }
58      CS->KEY = LOCK_CS_KEY;
59  }
```

Listing C.4: MSP432 Receiver Clock Header File

```
#ifndef __CLOCK_H__
#define __CLOCK_H__

#include "msp.h"

#define LOCK_CS_KEY 0
#define CLK_PER_LOOP 4

#define FREQ_1_5_MHZ 0
#define FREQ_3_MHZ 1
#define FREQ_6_MHZ 2
#define FREQ_12_MHZ 3
#define FREQ_24_MHZ 4
#define FREQ_48_MHZ 5

// Initialize DCO and other clock values
void init_dco(void);

// Set DCO to provide frequency
void set_dco(unsigned int freq);

#endif
```