Steering Sound with a Phased Speaker Array M.Eng. Report

A Design Project Report
Presented to the School of Electrical and Computer Engineering of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering, Electrical and Computer Engineering

Submitted by
Christopher Mehsen Bakhos
M.Eng. Field Advisor: Dr. Van Hunter Adams
Degree Date: May 2024

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering
Cornell University
Design Project Report

Project Title: Steering Sound with a Phased Speaker Array

Author: Christopher Mehsen Bakhos

Abstract: This M.Eng. project involves designing a device with no moving parts that allows for the user to steer sound emitted from an array of speakers. It does so by driving those speakers as a phased array, in which the relative phase from each speaker is carefully controlled such that the sound waves constructively interfere in the desired direction. The development process of this project was fairly straightforward throughout most of the project. An incremental approach was taken to develop this project over the course of the semester. Development began with programming a single speaker to emit a sound. After unit-testing this capability, more speakers were added to the system and an oscilloscope was used to confirm that the phase of the sound sent to each could be independently controlled. With this ability, that phase was carefully computed in order to generate directional audio. The resulting device steers sound without moving the speakers or any other components.

Executive Summary:

Steerable sound likely has some unique and creative applications. One usage of steerable sound might be in a concert setting, where some parts of the crowd can be made to hear one thing more prominently, while another part of the audience can hear another thing more prominently. Another possible application is if there is a sufficient range of frequencies, data could be encoded in those frequencies and used to communicate to a specific receiver, for some sort of targeted, wireless communication.

The phased speaker array worked as hoped. It was able to direct sound in a desired direction simply by changing the relative phases between speakers. The difference was obvious to listeners and measurable by devices. It satisfied the design problem.

One difficulty with development was solving for the relative phases for each speaker. One of the tools used to model this speaker array, MATLAB's Phased Array System Toolbox, did not state the relative phase offsets. This required separate modeling in Desmos 3D, which was quite simplistic compared to the Phased Array System Toolbox. Due to this, I was not fully confident that the numbers derived from this Desmos model were realistic and would translate well to reality. During testing, seeing if the phase was present was simply done by measuring the signal outputs with an oscilloscope. However, seeing if the phase interacted in the desired way could only really be tested for by observing the constructive and destructive interference patterns, which required the remainder of the system to be completed and implemented correctly. There seemed to be no way to test the phase offset functionality independently.

The methods used for testing the phased speaker array and collecting data from it were fairly ill-developed. Testing and data collection happened toward the very end of the semester, so not much thought was given to these aspects of the project. So beyond the first collection of data, there was no time to reattempt it or refine the methods. Reflecting on this, one interesting and useful thing to potentially pursue in the future would be to design a setup that could more rigorously record and measure the decibel data coming from the speaker array. Ideally, it would be able to create something akin to a heat map, where the loudest regions and softest regions from the speaker array would be made easily visible. This could be incredibly revealing about the state of the speaker array, and offer great insight on where it could use some improvement. And even just from a curiosity perspective, to see a heat map generated from the speaker array would be quite interesting.

Design Problem:

Over the course of the past year, this design project has evolved quite a bit. The original concept of the project and its requirements was that it would be a device that emits sound from some sort of speaker, or uses some physical apparatus to generate a noise, and listens for the returning echo. It would then be able to gather certain information about the surrounding area from this echo. The information gathered would somehow be broken down and processed, though it was never quite clear how this would be done. This processed information would then be remapped to the sense of touch, presumably using something like a collection of vibrating motors on the user's person, so they can feel the vibrations that carry the remapped spatial echo information. Over time, the user of this device would ideally be able to learn how to passively interpret what these vibrations mean in terms of their surroundings, and therefore would be able to obtain some intuitive understanding of their surroundings.

Toward the end of last semester, Hunter and I had begun discussing switching the project to something a little different. The idea of triangulating sound was discussed, where there would be stationary sound emitters in a room, set to emit different frequencies. There would be a microphone anywhere in this room that would pick up on these sounds. This microphone would then be able to tell when sound arrived at it, and from which emitter it came. With this, the system would be able to figure out its relative distance to each speaker, and its location in that space.

At the start of this semester, Hunter and I again discussed switching the project to something else. We kept the overarching theme of sound, except we steered the project more towards being able to direct sound. The idea we were trying to do was one where we would assemble an array of speakers to try to direct sound using interference, without physically pointing the speakers where we want the sound to go. We decided to go with this and ultimately settled on the design problem of getting a working demonstration of a phased speaker array.

Switching projects allowed me to be able to leverage much of the learning I had already done for the previous project idea, as the goal was simply modified.

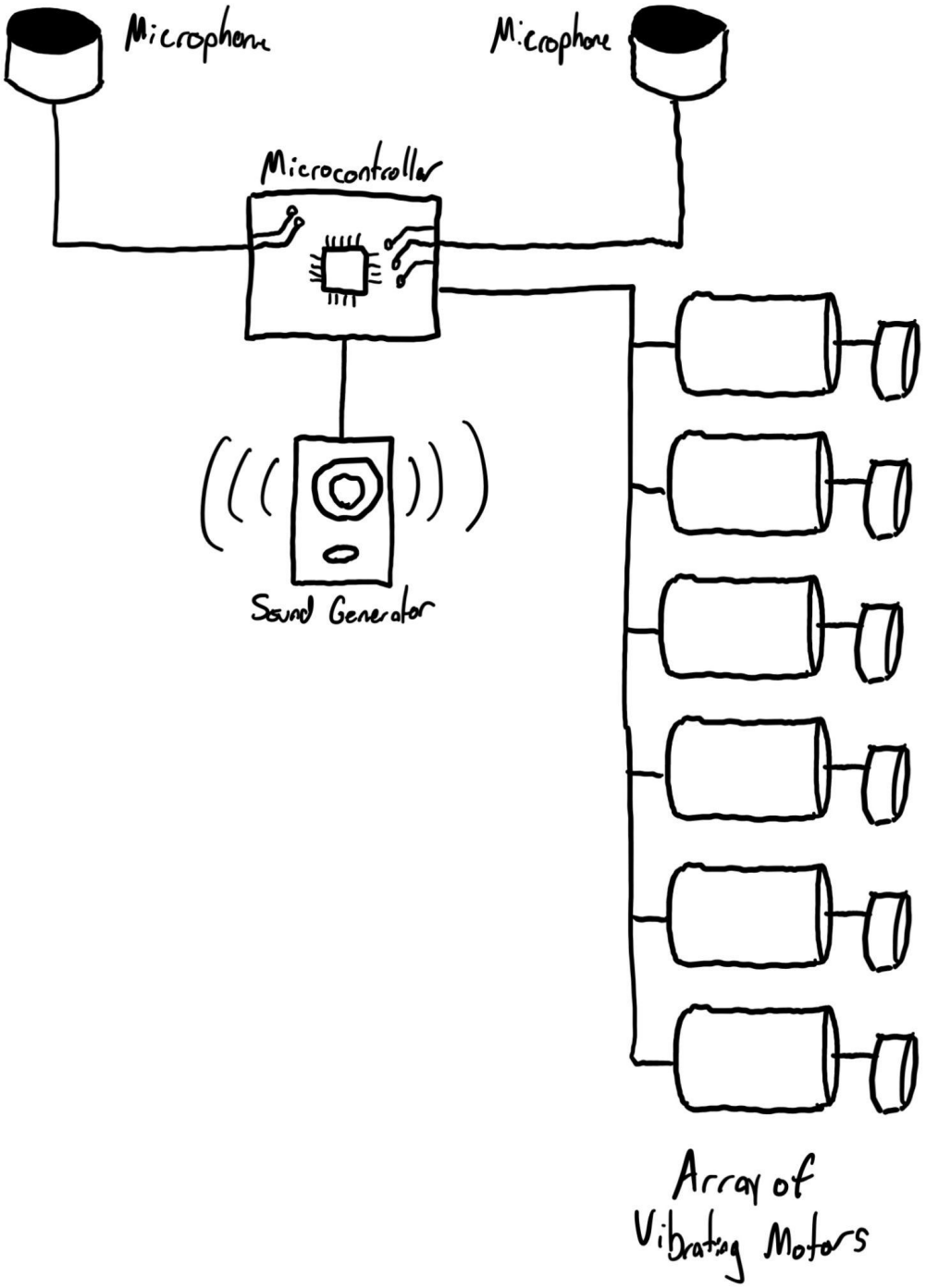

Figure 1. Diagram showing my conception of what elements the echolocation device might have had.

System Requirements:

The system requirement for the device was to perceptibly steer sound with no moving parts. Though this wasn't explicitly defined at the beginning of development, what this

implies is that the difference must be at least 1 dB if any arbitrary frequency is to be chosen. According to the paper "Lateralization of High-Frequency Tones" by A. W. Mills, the just-noticeable difference for sound around 1 kHz is about 1 dB, and it is lower for other frequencies[1]. So if a different frequency other than 1 kHz is to be chosen, then a lower decibel limit could be achieved, such as 0.6 dB with 1.5 kHz. However, a difference of 1 dB or lower could easily be caused by error. Therefore, a larger difference of 5 dB would both allow the difference to both be perceptible and robust against noise.

Another essential system requirement for this speaker array would be to be able to change during runtime the angle at which the sound is being emitted loudest. This is important, as emitting only in one direction mostly defeats the purpose of actually directing sound.

Another system requirement is that the sound generated must have a reasonably high synthesis rate. While this does not strictly follow from the perceptibility of the steered sound, a synthesis rate of 44.1 kHz allows the generated sound to express the full-range of human hearing frequencies, due to the Nyquist-Shannon sampling theorem. Beyond potentially calculating and setting the phase angle, the computation for this audio synthesis is very minimal, which should allow for this synthesis rate to be achievable.

Though changing the frequencies changes how the interference pattern works, and therefore would necessitate that the phase offsets and speaker spacing be different, on a small range of frequencies, this effect would be negligible. Therefore, the speaker array should be able to operate in a range of frequencies of 50 Hz of the chosen frequency. While this also does not strictly follow from the perceptibility of sound requirement, this would allow for at least some variety of sounds to play, and would allow the system to have a greater variety of uses beyond just generating the same beep repeatedly.

So, the phased speaker array, ideally, should be able to satisfy these requirements:

- Perceptibly steer sound with no moving parts
- Runtime phase adjustment
- Have a measurable and meaningful sound difference of 5 dB
- Have a minimum 44.1 kHz synthesis rate
- Have a minimum 50 Hz frequency range

[1] Mills, A. W. (1960). Lateralization of high-frequency tones. The Journal of the Acoustical Society of America, 32(1), 132–134. https://doi.org/10.1121/1.1907864

Review of Previous Work:

The main work I referred to was Hunter's birdsong lab[2] and his Raspberry Pi Pico audio demos[3].

The birdsong lab included relevant information on how to wire the Pi Pico to the digital-to-analog converter (DAC). It linked to various resources to aid me with the development of this project. These resources were invaluable to figure out the build environment, figure out how to use the DAC with the MCP4822 datasheet[4], and better understand direct digital synthesis (DDS), which is the mechanism used to generate sound in this project.

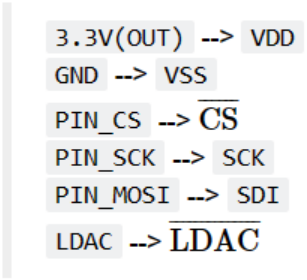

Figure 2. The mapping from the Raspberry Pi Pico to the MCP4822.

An even bigger foundation for this project was Hunter's audio demos. The code that drives the phased speaker array consists of modifications to Hunter's demo code. The first demo referenced was his single-core beep synthesis, where he used one core on the Pi Pico to synthesize beeps out of a single DAC channel. The other demo was his multi-core beep synthesis, where he then used both cores of the Pi Pico to output sound out of both DAC channels. This multicore demo ultimately became the basis of this project's code.

There were a few other resources that were referred to as reference more than for development. Ed Szoka and Tom Jackson had previously worked on a twelve-speaker

---

[2] Adams, V. H. (n.d.). Synthesizing birdsong with the RP2040. Van Hunter Adams. https://vanhunteradams.com/Pico/Birds/Birdsong.html

[3] Adams, V. H. (2023, June 8). Audio synthesis and processing examples of incremental complexity. GitHub. https://github.com/vha3/Hunter-Adams-RP2040-Demos/tree/master/Audio

[4] https://vanhunteradams.com/Pico/Birds/DAC.pdf

phased speaker array as their final project for ECE 4760[5]. The Wikipedia page for phased array[6], as well as the video "What Are Phased Arrays" by MATLAB on YouTube[7] were also quite useful in getting a general understanding about phased arrays. The video introduced me to MATLAB's Phased Array System Toolbox[8], which proved to be a valuable resource for development as well.

Range of Solutions:

Being that this design problem was making a phased speaker array, there was a wide range of solutions, even if the basic concept had to be the same. By definition, speakers were required for this design problem. However, the kinds and number of speakers could vary. One option was the desktop speakers in the lab. These aren't very easily maneuvered and are quite bulky, so a reasonable number of those would be on the order of eight or fewer speakers. Also easily available were the small piezo speakers. These seemed to be a lot smaller and more manageable, so having on the order of sixteen or so speakers seems more reasonable. Both of these were limited by how many speakers the microprocessor could be programmed to drive. It is important to keep in mind when discussing the range of possible solutions, that choosing a different number of speakers would possibly inform the decision of what frequency, or range of frequencies, could be used to drive the speakers. And the spacing between the speakers would likely have to change as well.

Another possible factor to consider regarding solutions would be whether or not the speakers would be configured in a 1D array or a 2D array. A 1D array arranged left-to-right would allow for steering sound left and right, but not up and down. However, a 2D array would allow for sound to be steered both left and right, and up and down.

Most microcontrollers would have likely been sufficient to run the code that will properly drive the speaker array. The microcontroller simply needed the ability to perform an interrupt every interval of time and communicate over a serial peripheral interface (SPI).

---

[5] Szoka, E., & Jackson, T. (2012). ECE 4760 Final Project: Phased Array Speaker System. Cornell University School of Electrical and Computer Engineering. https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/tcj26_ecs227/tcj26_ecs227/index.html

[6] Wikimedia Foundation. (2023, November 26). Phased array. Wikipedia. https://en.wikipedia.org/wiki/Phased_array

[7] MATLAB. (2022). What Are Phased Arrays? YouTube. YouTube. Retrieved December 14, 2023, from https://www.youtube.com/watch?v=9WxWunoE-PM.

[8] https://www.mathworks.com/products/phased-array.html

Both of these things are fairly common in microcontrollers. And there is even the option of implementing the SPI protocol in software via bitbanging, if dedicated SPI ports were not available.

Many options for digital-to-analog converters (DACs) would work. Being that there would be an SPI channel from the microcontroller to communicate the data to the DAC, a DAC that accepts SPI input would certainly be simpler to work with. However, there are likely workarounds if an SPI DAC were not to be available.

In terms of generating phase differences between the speakers, there seems to be two main ways to do that. The first is to emit all the sounds at the same time, but offset the phases in the sound generation by choosing a different point in time to sample from the sound being produced. The other way is to keep the phases the same, but emit the sounds at different moments in time, effectively changing the relative phase offsets as the observer experiences them.

All these solutions create a wide array of possible designs for the speaker array.

Selected Approach:

The design choices regarding the components used for this implementation were mostly due to their ease of use. This ease lies in that they were all easily available in the lab, with no need to order and wait for specific parts and that there were good references, so development was easier.

While both the desktop speakers and the piezo speakers were readily available in the lab, the desktop speakers in the Phillips 238 lab were used in this implementation of the phased speaker array. These speakers are relatively straightforward, with minimal setup required. Simply plugging them in and turning their volume up will allow them to output whatever data they receive in their aux input. They are therefore very simple to get working. These speakers, however, would certainly be less portable, less customizable, less self-contained, and less from-scratch. Though I didn't realize this at the time the calibration of these speakers is a lot less precise, as the speakers have knobs that are difficult to turn to the same position for all the pairs of speakers being used. Ultimately, the simplicity of the desktop speakers outweighed the flexibility of the piezo speakers.

I opted to design a system using a 1D array of speakers. It would likely take a significant amount more work to get a 2D array of speakers set up, and as a proof-of-concept demonstration, a 1D speaker array would be sufficient to satisfy the aim of this project. Another thing to consider is that the desktop speakers also would not have been optimal, either. Their shape would not allow them to be easily stacked on top of each other. And so for this, piezo speakers would have been needed. A frame to hold all of them in the proper place with the proper spacing would have had to have been constructed. So even though designing a 2D array was not feasible to start off, future work could certainly explore expanding the project to be a 2D array.

For the microcontroller, the Raspberry Pi Pico was used. The Pico had all the computing power, memory, serial interfaces, and other features needed to drive the speakers. Along with this, it was easily available, there were many demos Hunter had developed for it, and it was what Hunter had been using for his ECE 4760 labs, which the birdsong lab was a part of. Being that the birdsong lab and his audio demos were so strongly related to this project, using a different microcontroller and starting from scratch would have been needlessly more difficult and would have led to unnecessary extra work.

The same applied to using the MCP4822 for the DAC. It was easily available, and was already used in Hunter's audio synthesis demo. It had an easy SPI port interface, which made data transfer very easy.

In terms of generating the phase offset, I chose to have the speakers emit all the sounds at the same time. This is a bit nuanced, so it is important to first understand the algorithm used to generate sound, direct digital synthesis (DDS). This algorithm uses overflow in variables to create cycles, as a variable will go from its smallest value, to zero, to its biggest value, and wrap back around to its smallest value upon overflow. These cycle variables allow for indexing into a precalculated sine table, instead of calculating complicated math during runtime so as to save CPU time and allows for generating sounds with a high synthesis rate. The variables indexing into this sine table can be manipulated to allow for easy and precise control of phase offset. The sine values are used to calculate audio data to send to the DACs. The DACs will hold onto the data they are given until one of their pins, the Latch DAC Input (LDAC), is held low for a short time. Connecting all the LDAC pins together on all the DACs allows for the DACs to release all of their sounds to the speakers at the exact same time. This also simplifies design and helps ensure no additional timing delays (and therefore unwanted phase

offset) from the program execution sending the LDAC signals sequentially via GPIO. This is all much simpler than trying to time the speakers differently, as adjusting timing can often be finicky and less precise.

If I were to continue development of this phased speaker array, I would consider taking it in the direction of using the smaller piezo speakers to see if I could make a larger array, and to see if it could possibly be expanded to two dimensions. Also, if development were to continue, there would likely have to be additional considerations for the microcontroller to be able to drive many more speakers. Perhaps a different microcontroller, more microcontrollers, or some external logic might be needed. However, for what this M.Eng. project was aimed to accomplish, the choices made were the best. As it stands now, it is not exactly the most expandable or flexible, however, it was simple enough to get a working demonstration.

Design and Implementation:

After discussing the project idea, an incremental approach to designing this phased speaker array system was decided upon. The first step was to try to use Hunter's Pico demo code that synthesizes a beep on a single core to generate a sound using DDS. This required the DAC to be hooked up to the proper pins on the Raspberry Pi Pico, as well as the aux port being hooked up to the proper pins on the DAC (refer back to figure 2). For this step, I used a pair of earbuds to be able to tell if sound was being produced.

The next step beyond getting Hunter's single core code to generate a sound was to get sound to come out of both speakers. Hunter had done this by leveraging the Pico's multicore. In trying to run this, a build issue started occurring. This was a bit perplexing, as this error did not seem to be a build error at first. Regrettably, I did not record what the error was, and it's likely not duplicatable, as the old build environment was completely removed from my computer to try to fix it. Installing the build environment manually, instead of using an auto-installer seemed to fix the issue.

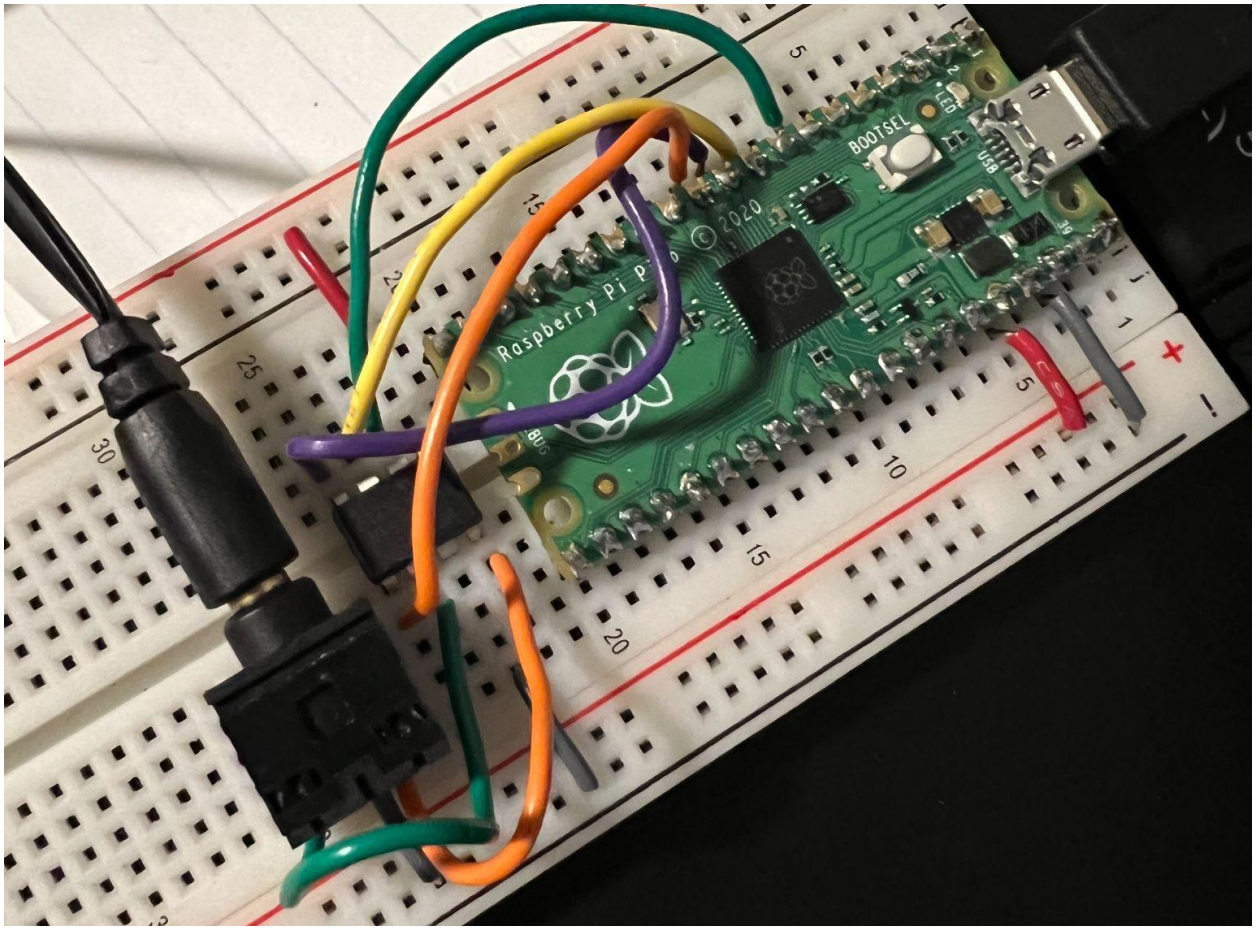

Figure 3. An image showing the Raspberry Pi Pico connected to the MCP4822 DAC and an AUX port. The grey wires are ground and the red wires are 3.3V.

Once this was solved, the next step was trying to change the timing of the beeps coming out of the speakers. To simply try to delay the beeps between the two speakers was the goal, and this was pretty simple to achieve. It involved simply placing a sleep statement after launching core 1 and before scheduling core 0 threads. This made it so that there was always this offset between the two core threads that continued during the remainder of the program execution.

Subsequently, to be able to control the delay with a certain level of precision seemed like a logical next step to getting a phased speaker array. The mini demo to be able to demonstrate that this was possible was to generate sound revolving around the head of the listener, by adjusting the relative delays of the beeps. If the two beeps were generated at the same time, they should give the sense of coming directly from the front of the listener. If the beep on the left came before the beep on the right, the listener should, in

theory, get the sense that the beep was coming from the left. The illusion of directional sound was not so important, but rather the timing being controlled was the most important aspect of this. Because of this, there was a lot of oversimplification in this step. I assumed the distance from 1 ear drum to another is 1 foot apart. I also decided that sound was going to be generated as if it was somewhere on a circle 1 foot around the center of the head. With this, I was able to build two delay tables. The values of the delay tables were calculated using the distance between ears, the radius of the circle, the angle, as well as the speed of sound. These each held the values for eight points on the circle. For each sound generation, these two tables were accessed and their difference was used to choose which earbud to send a sound first, and how long to wait before sending the sound to the second earbud. This seemed to work, though it was a bit difficult to tell. Increasing the delay by a factor of 1000 made it very clear that there was indeed a delay present. However, using an oscilloscope was the way it was precisely verified.

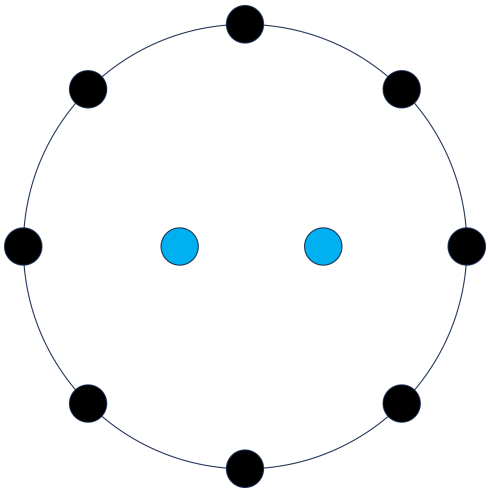

Figure 4. A rudimentary diagram showing the placement of the eight points on the circle, represented by black dots, surrounding the two ears, which are represented by the blue dots.

Then after this was functional, the next step Hunter and I decided upon was to try to get this coming out of two SPI channels. This was rather simple, as the logic that went into generating the sound for the previous exercise could just be duplicated. And since the Pico has at least two SPI channels already configured from the start, it was rather easy to be able to just output the duplicated data to that SPI channel. There was no need for additional wiring compared to what was already done. Simply testing that the outputs from the second SPI channel were as expected was how correctness was ensured. This

simply required rewiring the MOSI, CS, and SCK pins of the DAC to the second SPI channel's pins from the first SPI channel's pins.

Instead of starting the speakers at different moments in time, we realized we really wanted to adjust the phase of the waves, which allows more precision anyway and is simpler to implement. With this realization, the next step was to actually change the phase of the outputs and keep the timing the same for the sounds. It was simple in concept to just start the sine table index variable having an offset to begin with, and remove the code that changes the timing.

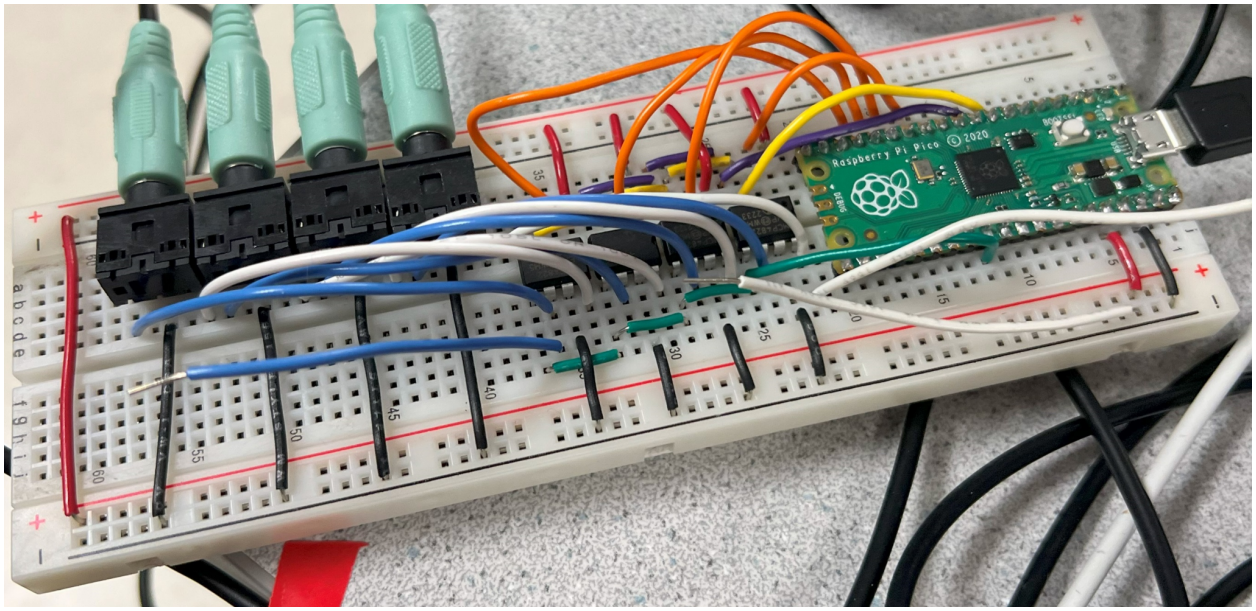

Figure 5. A picture of the breadboard as it was wired during my demos and data collection. The black wires are ground and the red wires are 3.3V. The yellow wires are the MOSI lines for each DAC, the purple wires are the SCK lines for each DAC, the orange wires are the chip select lines for each DAC, and the green wires are the LDAC signal for all the DACS. Lastly, the white wires are connected to each DAC's channel A, connecting to the right speaker and the blue wires are connected to each DAC's channel B, connecting to the left speaker.

After doing this, the next step was to set up the code so this would output to four different DACs, for eight different speakers. In making the move toward eight speakers, continuing the pattern of one SPI channel per DAC would have been a difficult one, as configuring the Pi Pico to have four SPI channels is unnecessarily difficult. The workaround to this is much simpler and more elegant than configuring four SPI channels and hooking up one to each DAC. I simply needed to hook all the DACs up to

the one SPI channel. However, the chip selects needed to be manually controlled, though, as the chip selects had to be different for each of the four DACs. The chip select lines are used to inform their respective DAC whether or not they are to pay attention to their input, effectively allowing a way to select which DAC to use. All of the LDAC pins of the DACs were connected to one GPIO pin of the Pico, allowing the LDAC pin to be pulled down when the sounds needed to be sent to the speakers from the DACs, all at once. This allowed for the sounds to be sent synchronously, without any additional delays with trying to deal with four separate LDACs.

The last big problem, though, was figuring out how the desired angle to direct the sound related mathematically to the phase. The frequency needed to be accounted for as well, as that changes how the phase relates to the angle. To figure this out, started trying to model the situation in 3D Desmos. The first step was to model the sound waves as a sine wave scaled by the distance from the sound's origin.

$$sin(f * \sqrt{x^2 + y^2}) * \frac{1}{\sqrt{x^2+y^2}}$$

The sine wave represents the wave, and the inverse-distance term represents how the loudness (or quite frankly anything comparable to that, we just want to model the sound as decaying over a distance) of the sound wave falls off at a rate of one over the distance.

This, however, only represents the sound coming from one speaker, and not an array of speakers. By adding an offset term to x, we can start to model an array of speakers, with each speaker incrementing a by 1.

$$sin(f * \sqrt{(x - a)^2 + y^2}) * \frac{1}{\sqrt{(x-a)^2+y^2}}$$

Now, all of these speakers in this array are simply outputting the exact same sound wave, just offset in space. Accounting for constructive interference, this would result in the sound always being loudest directly in front of the middle speakers. To be able to steer this sound, however, a phase term needs to be added. This is then the final equation for each individual speaker.

$$sin(f * \sqrt{(x - a)^2 + y^2} + b) * \frac{1}{\sqrt{(x-a)^2+y^2}}$$

And so to actually have a graph of the constructive interference and destructive interference, we need to add all of the waves together, from all speakers, 0 to N.

$$\sum_{a=0}^{N} sin(f * \sqrt{(x - a)^2 + y^2} + b) * \frac{1}{\sqrt{(x-a)^2+y^2}}$$

Being that Desmos 3D is being used to solve for the phases of this array, solving for the term b is the main purpose here. From graphing the previous equation and from intuition, when the b value for each speaker is 0, the constructive interference is strongest and directed straight ahead of the speakers. A fair first assumption would be that b is evenly spaced from one speaker to another, within a certain range of phases. Building off this assumption, the following is the equation for the value of b, as dependent on speaker a and angle θ.

$$b(a, \theta) = (\frac{2\pi}{N-1} * a - \pi) * (-\frac{2}{\pi} * \theta)$$

This is really just a linear equation with respect to a. Looking at the terms within the first set of parentheses, we can see that as a increases, from 0 to N-1, the value goes from -π to π, at intervals of 2π/(N-1). This satisfies the evenly spaced assumption. The assumption that there would be a range of phases is satisfied by the range -π to π. This term must be present so as to have the equation be dependent on a. The phase must be dependent on the speaker, as that is the whole mechanism behind a phased speaker array. The second set of parentheses can be thought of as characterizing another linear equation, this time, with respect to θ. As θ increases, the value of b decreases by a factor of 2/π. This term must be present because the phases must be dependent on angle. This equation gives the phase angles needed to be able to direct the sound.

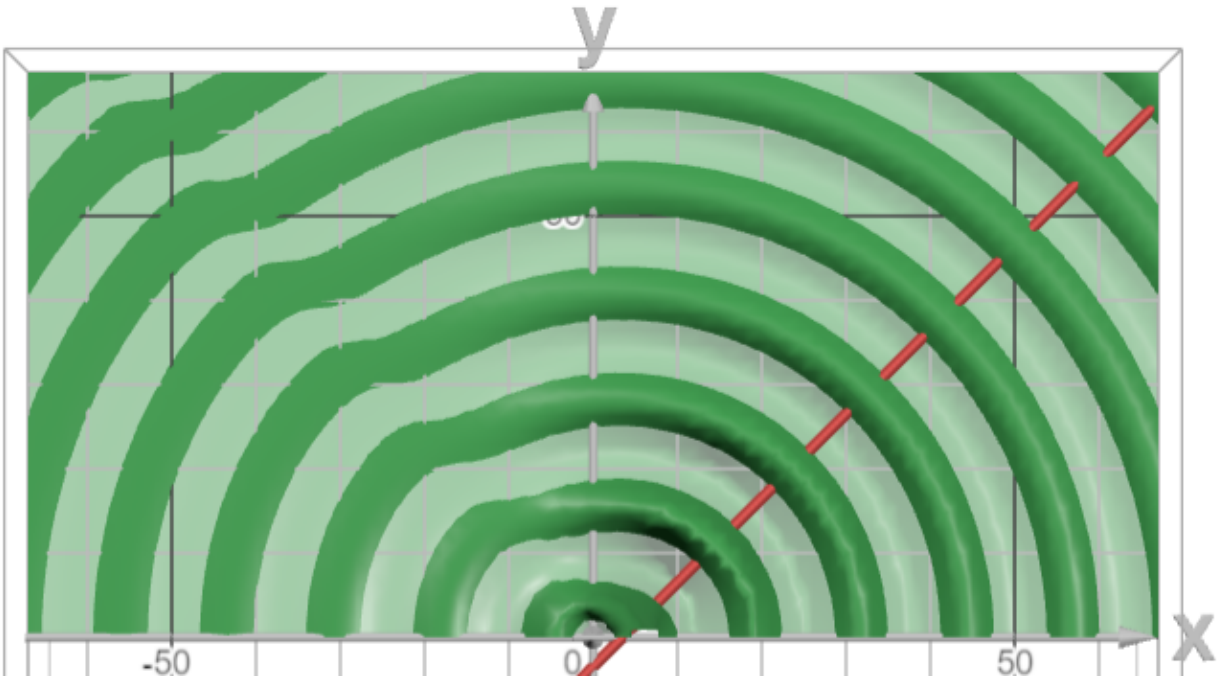

Figure 6. A top-down view of the 3D Desmos graph I created to attempt to model sound waves traveling in a plane, where vertical amplitude (only visible indirectly due to shading, here), represents loudness. The red line represents the direction the sound is supposed to travel (toward the top-right of the image, at a 45° angle, from the center of the speakers). The amplitude is largest along this red line.

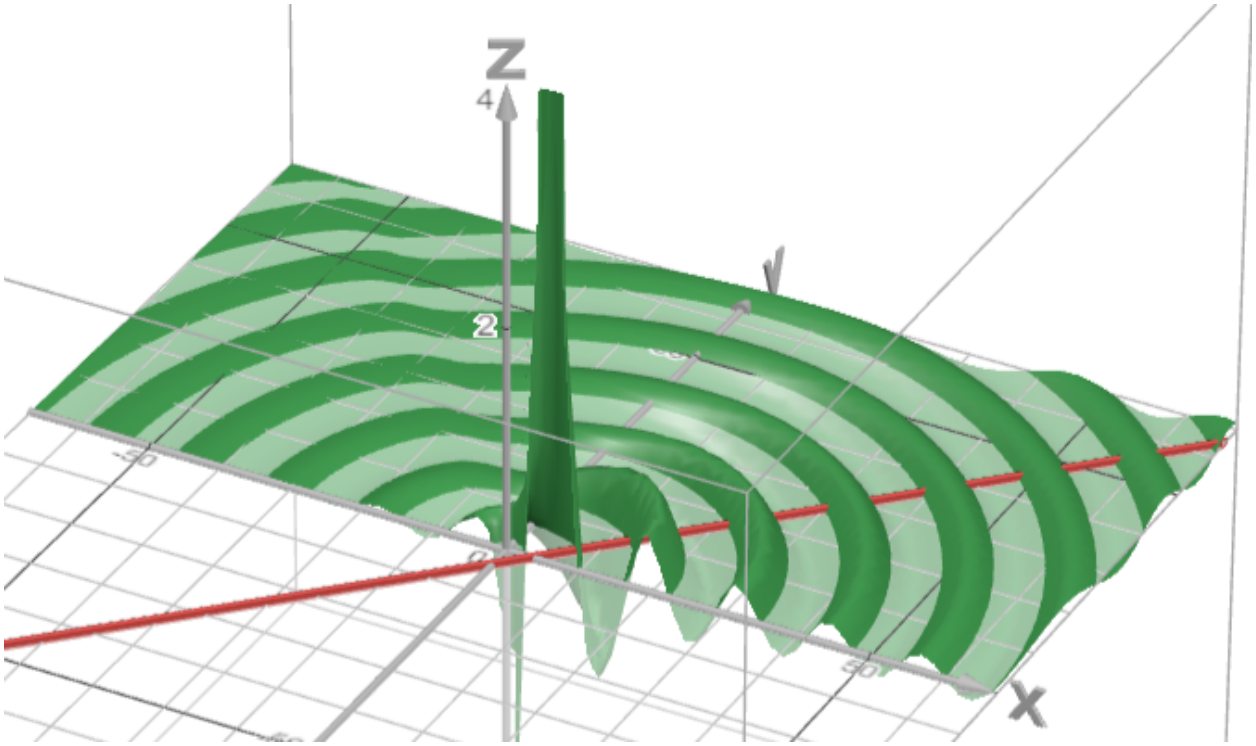

Figure 7. An angled view of the same 3D Desmos graph. The vertical amplitude, representing the loudness of the speakers, is more visible here, and it can be seen that the graph is generally reaching maxima along the red line.

The MATLAB Phased Array System Toolbox helped in figuring out the proper frequency that would allow a good range of direction, given the number of speakers one Pico can reasonably drive, as well as the spacing between each speaker. With one Pi Pico, it is reasonably simple to drive eight speakers, and though that can certainly be pushed to higher numbers, given time constraints, effort was better placed elsewhere. The spacing between the speakers was also a concern, and given that there are eight speakers, a reasonable choice for the spacing is on the scale of half of a wavelength. This turns out to be around 4.5 inches for a frequency of 1500 Hz. Much closer together than 4.5 inches would be impossible for the bulkier desktop speakers. And much further away would be infeasible, as they would be quite far apart.

With all of these variables in place, I could set the phases in a lookup table to save computation time during execution. One of the system requirements for this speaker array is the ability to adjust phase during runtime. The way the sound is generated is that there is a while(true) loop, constantly running until the microcontroller is turned off. Within that while loop, there is another while loop that waits for a global variable,

count, to be less than a value BEEP_DURATION (10400 loop iterations, or 0.26 seconds, in my current implementation). The variable count is manipulated in a repeating timer callback that happens every 25 µs, for 40000 times per second. This timer callback is what actually generates the sound. It will increment the value that indexes into the sine table so we can grab the next value for each speaker. It then, for each speaker, actually indexes into the sine table, pulls down the chip select, outputs the data to the DAC, and pulls up the chip select. Then, it increments the count variable. Lastly, it pulls down the LDAC, waits at least 100 ns (but in the case of this implementation, 1 µs, because it can't wait any shorter than that), and pulls up the LDAC to release all the DAC data to the speakers. That is what outputs the sound. After enough of these timer interrupts happen to increment the count variable to be equal to BEEP_DURATION, it can then change the phase before the next sound is generated. It then will sleep for a measure of time, before setting the count to zero so the process can begin again.

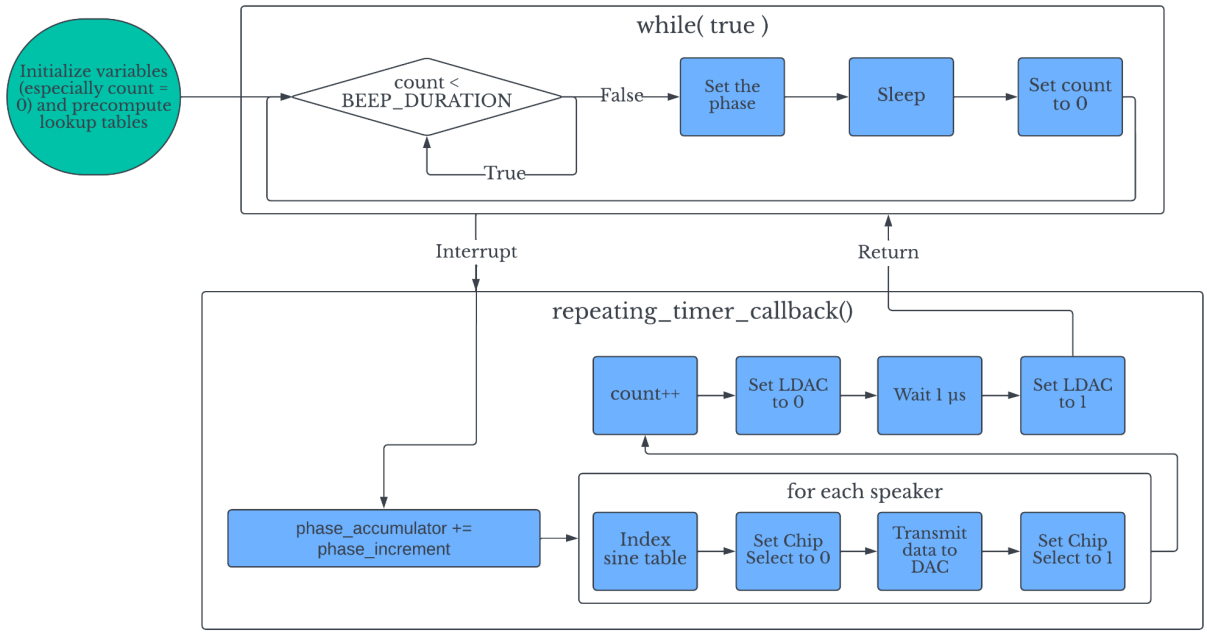

Figure 8. A flowchart showing how the program that drives the speaker array progresses through its execution. This visualizes what was described in the paragraph above.

Once all of this is said and done, I hooked it up to four speakers to see if it actually worked. I taped red tape to the knobs to keep track of their position, as they had no real way to tell their volume with certainty unless they were off or at full volume, both of which were not reasonable.

Figure 9. An image, showing off all eight speakers of the speaker array. The red tape on the right speakers of each pair of speakers was used to keep all the volumes of the speaker pairs all the same.

I made sure all the speakers were approximately 4.5 inches apart, though there was a point where I could only estimate this.

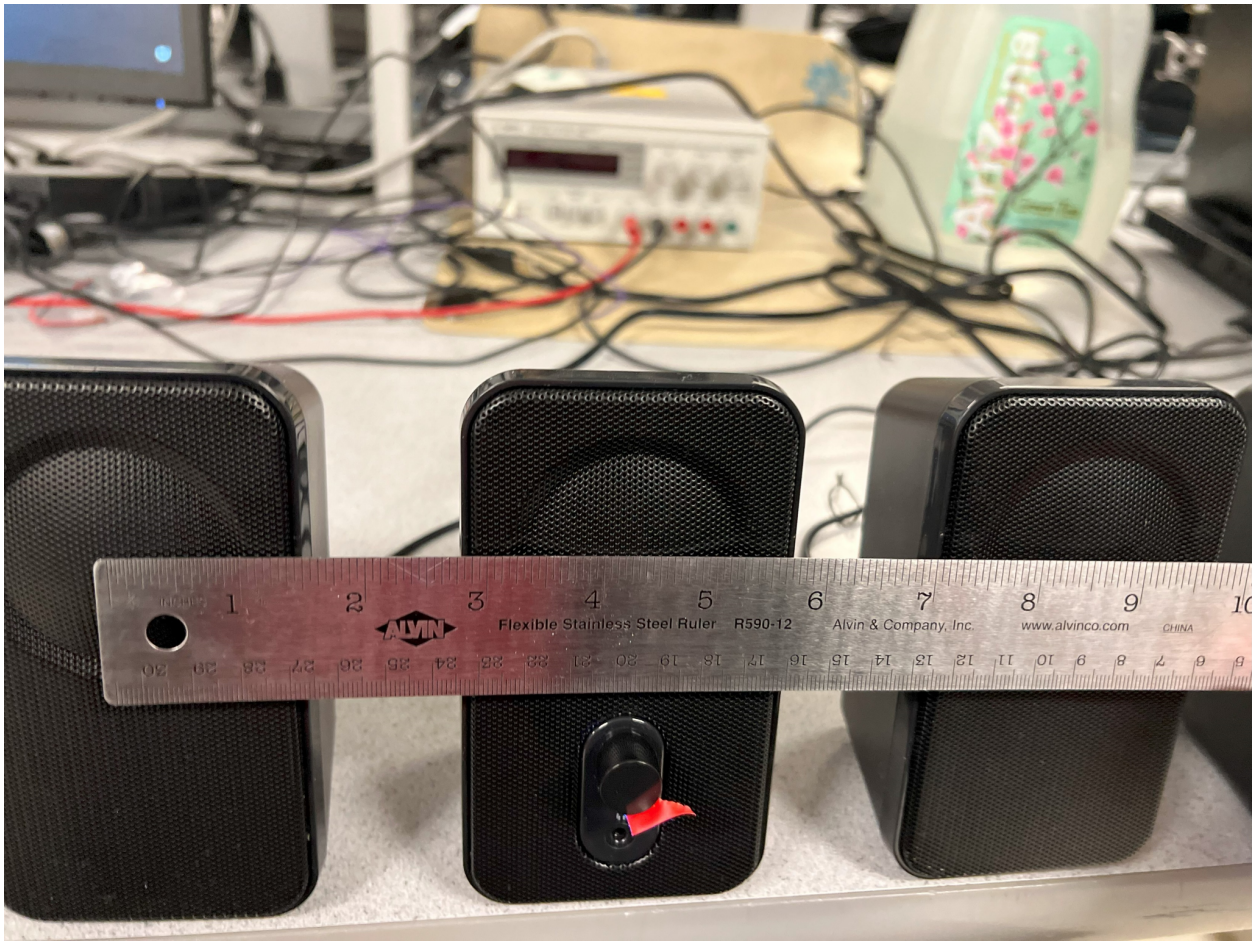

Figure 10. A ruler showing the spacing between the speakers to be around 4.5 inches. This also provides a close up of the red tape used to keep track of the volume of a pair of speakers. The tape was placed at the top when the volume was all the way off.

System Test Results and Verification:

In talking to Hunter about our expectations for the array (we actually had discussed this after I had already tested it), we both expressed that we were not really all that confident that it would work. Based on the calculations and simulations, everything seemed like it should work. The MATLAB simulation made it seem like the numbers for the frequency, the count of speakers, and the spacing were reasonable and would yield good results. And the Desmos graph also made it seem like I had chosen proper values for the phase, and that would work, too. But admittedly, I wasn't too confident that my Desmos graph was right, and therefore that my phases were correct either. I also wasn't too confident that the phase offset for each individual speaker was being properly output as was theoretically expected, as looking at an oscilloscope reading of the output, it didn't always seem to come out correctly, even if it did most of the time. Beyond the doubts, my expectation, if it were to work, would have been along the lines of the effect being perceptible, though not well, and inconsistent.

To initially test whether or not the phased speaker array worked as desired, I set up the speaker array and just tried to listen to any audible differences. Then the next step, beyond just qualitatively seeing if it worked, was measuring if it worked. I did this very rudimentarily. I used my iPhone, on which I had downloaded a decibel meter app. I used the microphone connected to the set of earbuds that came with the phone to actually pick up the sound. I realized I needed to measure at a consistent distance, so as to get accurate results. With this, I used a ruler and dangled the microphone at the end of the ruler, giving readings up to about twelve inches away from the array.

Figure 11. An image showing how I held the microphone and ruler to collect the data. I needed a hand to take the picture, so in reality, I was dangling the microphone just a little bit above the end of the ruler.

Compared to what we would have expected from the array, it worked pretty well. The difference was perceptible, at least to Bruce, Hunter, and I. Originally, my demo was to point the sound in three directions, cycling the loudest sound from right to middle to left, when facing the speakers. This allowed the observer to stand in one of three directions and hear the sound loudly for one out of three pulses, and allowed the observer to move around and hear the difference in volume at each position. To better ensure it actually worked, I eventually switched to having the loudest sound directed 45° to the right when facing the speakers. This especially allowed for easier measurements, as it would be difficult to know which direction the sound was supposed to be pointing to at any given moment, the measuring setup was quite difficult to get set up at each measuring point, and I would want to take many data entries for each point, and the cycling sound would just make that much more difficult.

To try to get as much useful data as possible with my setup, I decided to take data at a variety of locations. I took data at each of the speakers to measure its loudness. This allows us to see how loud each speaker is relative to the other speakers, and it allows us to see to some extent how much the loudness at the other points of measurement were influenced by the relative loudness of the speakers. For example, if the speakers were significantly louder on the right side of the array, and the array emitted sound that was similarly louder on the right side, it would be difficult to tell how much of that loudness is caused by the phase offset as opposed to just the fact that the speakers on the right. When I was first demonstrating the array, when the loudest sound was cycling through, Hunter qualitatively tested each speaker to see if it was changing in volume with each pulse. If certain speakers were louder once every three pulses, that would show that the effect was not solely due to the phase offset. Regretfully, I did not test for this with a decibel meter, as that test, I believe, with a fair amount of confidence, proves that the effect is not due to relative speaker loudness, but rather to the phase offsets' effects. However, measuring that the speakers' loudnesses were fairly comparable and all fall within a small range of each other, that each speaker was fairly consistent across their measurements, and that the speaker's relative loudness does not seem to strongly affect the other measurements shows with fair confidence that it really is the phase offsets causing the measurable differences.

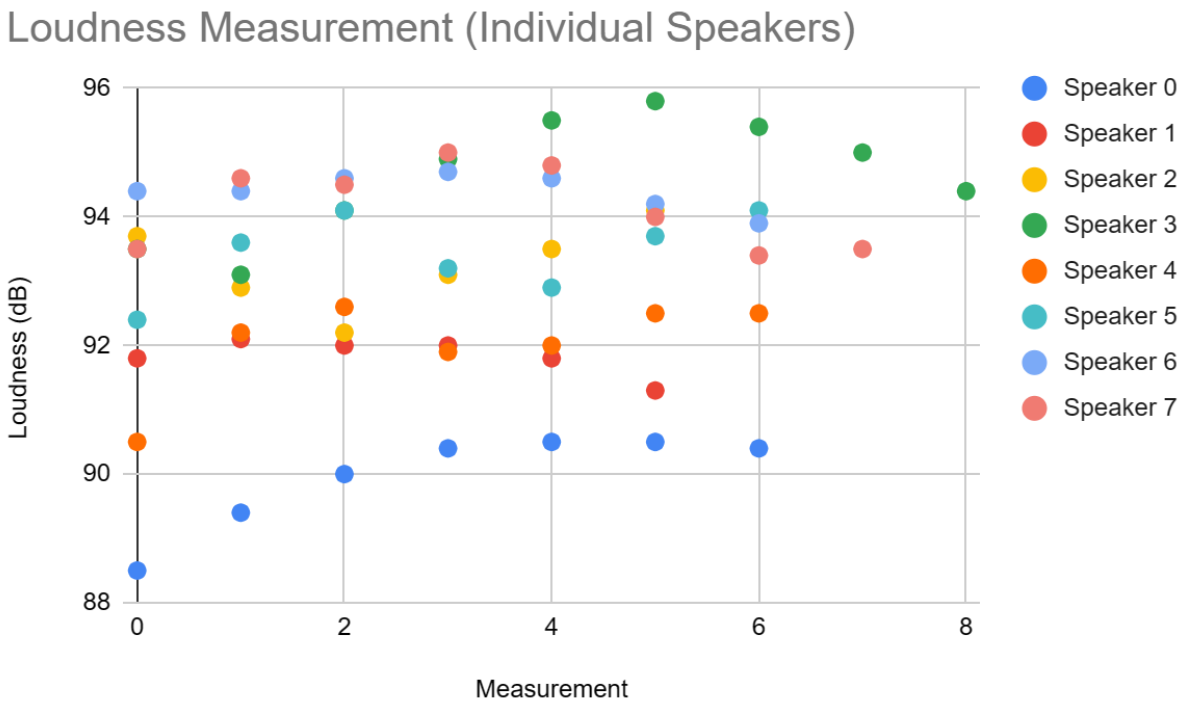

Figure 12. Graph showing the individual speaker measurements. The loudness axis is set to go from 88 dB to 96 dB. This makes it so that the individual measurements are more discernible.

Speaker	Average of Speaker Measurements (dB)	Variance of Speaker Measurements (dB)
Speaker 0	89.957	0.570
Speaker 1	91.833	0.083
Speaker 2	93.25	0.447
Speaker 3	94.633	0.86
Speaker 4	92.029	0.526
Speaker 5	93.429	0.399
Speaker 6	94.4	0.077
Speaker 7	94.163	0.414

Figure 13. Table showing the average of the speaker measurements and the variance of the speaker measurements for each speaker.

The table summarizes the findings from my data collection. The averages are relatively close together. The highest one being 94.633 dB and the lowest one being 89.957 dB. This makes for a difference of 4.676 dB. Though they are close together, still, this difference is admittedly and unfortunately non-negligible, and likely had an effect on the volume at the other measurement locations. The other set of values shown in the table is the variance of the measurements of each speaker, which shows that they are fairly consistent. And don't seem to change too much.

Figure 14. A picture of how I measured the loudness of each speaker.

Admittedly, only in writing this report do I realize that the reason the volume of the speakers were different was almost certainly to do with the knobs not being adjusted to

the same volume, as the red tape method was only an approximation. If I were to repeat this demonstration and the measurements, I would adjust the knobs to be as close as possible to one another by measuring the output from each speaker directly and adjusting.

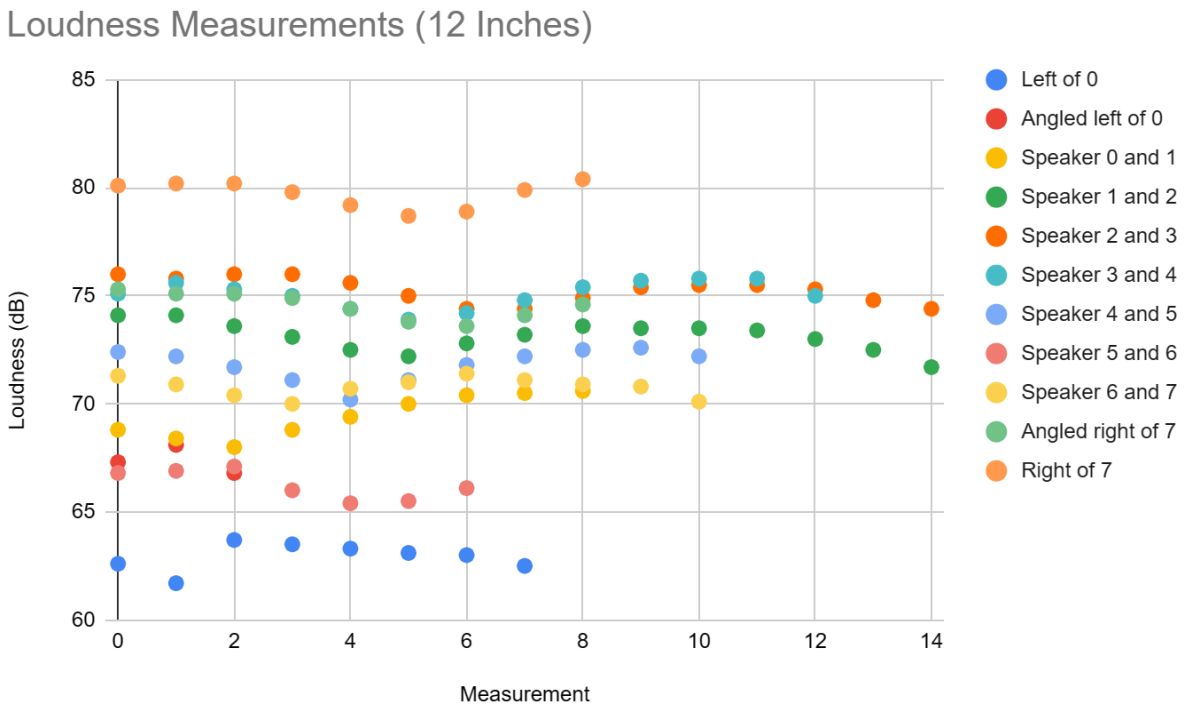

Figure 15. A graph showing the individual measurements at each location. The loudness axis goes from 60 dB to 85 dB to allow for seeing the smaller differences in measurements.

Measurement Location	**Average of Measurements (dB)**
Left of Speaker 0	62.925
Angled left of Speaker 0	67.4
In between Speaker 0 and Speaker 1	69.433
In between Speaker 1 and Speaker 2	73.12
In between Speaker 2 and Speaker 3	75.267
In between Speaker 3 and Speaker 4	75.077

In between Speaker 4 and Speaker 5	71.818
In between Speaker 5 and Speaker 6	66.257
In between Speaker 6 and Speaker 7	70.782
Angled right of Speaker 7	74.544
Right of Speaker 7	79.711

Figure 16. A table showing the average of the measurements at each location, twelve inches away from the speakers.

Something interesting to note is that between speaker 4 and speaker 5, which are 92.029 dB and 93.429 dB respectively, there is a measurement of 71.818 dB. However, between speaker 5 and speaker 6, which are 93.429 dB and 94.4 dB respectively, which overall are louder individually, have a much lower measurement in between them, of 66.257 dB. This might be near a point of destructive interference. This reasonably shows that the interference does have a measurable effect, as the speaker loudness alone cannot account for this measurement difference, as we would expect the measurement between speaker 5 and speaker 6 to be at least comparable, if not louder than the measurement between speaker 4 and speaker 5. However, it was significantly softer, so it could not have possibly been speaker loudness alone causing this effect. This might be a bit easier to see in the diagram below.

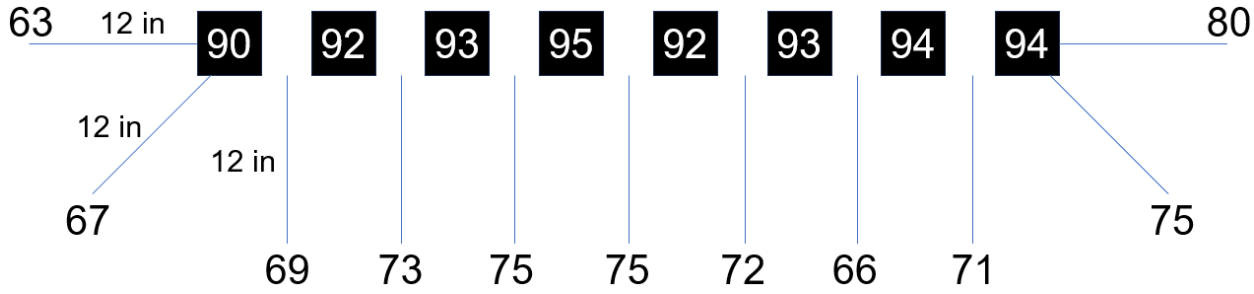

Figure 17. A diagram showing the decibel readings. The black boxes represent each speaker (0 to 7, left to right), with the number inside representing the decibel reading for that speaker. Each line represents the 12-inch distance and direction of each measurement, with the corresponding measurement. The readings are all averages for that corresponding area, and have been rounded to two significant figures.

This diagram can give a good visual representation to accompany the recorded data. It shows that, in general, that toward the right of the array, the output is louder and toward the left of the array, the output is softer.

In the future, if I wanted to better test it, I would set up a 2D array of microphones in front of the array of speakers, parallel to the floor, right below the speakers. These would ideally record the output of the array, or perhaps just the loudness of the array output. I could take this data and see how the array actually compares to any sort of theoretical calculations. It would be important to pay attention to the spacing and placement of the elements of this array. They would certainly need to be spaced at least at the proper resolution to be able to see the interference pattern. But doing this would give a really great idea of what the output of the speaker array is and how well it is working.