Acoustic Direction Finding with Arduinos
PHYS 398 Group 2 Report

Zhuo Chen,* Kaiwen Hu,* Leyang Liu,* Cheng Peng,* and Wenqing Wang*

(Dated: November 2020)

Locating the source of a sound signal is both interesting and very important in various applications. Achieving such a task using only Arduino single board computers and crude microphones can be very challenging. In this paper, we present an algorithm to perform reasonable predictions of the direction of the sound source. With only the access of sound recordings, we first use the L_2 metric (Euclidean distance) to extract the arrival times of the sound signal. Then, we define a cost function, which, when minimized, gives the predicted location of the source. We test our algorithm in various settings and show the robustness of our algorithm despite the simple recording devices. Our work provides a general algorithm to locate the source of a sound signal and can have a wide range of both civil and military applications.

I. INTRODUCTION

Acoustic point-source locating has been widely studied in the context of military and civil use. Acoustic direction sensors have a wide range of applications; from artillery sound ranging to mortar firing detection on battlefield [1] to gunshot detectors to crime monitoring [2]. Their detection methods can be categorized into active acoustic location and passive acoustic location [3]. Active acoustic location involves the creation of sound to produce echos, which can be used to locate the object. Passive acoustic location only involves the detection and analysis of the sound or vibration created by the object. [3] Military application of acoustic sensors mainly utilizes active acoustic location because the sound generated by aircraft or missiles are extremely weak over long distances. Civil applications focus on passive acoustic location, which requires less equipment and is much cheaper, but with limited detection range and accuracy.

The methods to realize passive acoustic location include but are not limit to Particle Velocity [4], Time Difference of Arrival [5], and Steered-Response Power Phase Transformation [6]. Particle Velocity exploits the fact that sound is conveyed through the vibration of air, and measuring the movements of air could obtain the direction of the sound source. An example of this method is the Microflown technology devised by the Microflown company [7]. Time Difference of Arrival method uses a sensor array to obtain the source direction by comparing each probe's signal. This is the most commonly applied method, and Urban Institute, a non-profit research organization, proposed a gunshot detection technology based on this method [8]. Steered-Response Power Phase Transformation is a novel algorithm using a beamforming approach that searches for candidate position that maximizes the output of a steered delay-and-sum beamformer. This method will also need a system of microphones and can operate in an adverse acoustic environment. Researchers have fully studied this algorithm and have obtained mature results [6].

Our project is based on the Time Difference of Arrival method and aims to make acoustic sensing technology available for ordinary people. We use Arduino-based sensors with data acquisition software to record sound and other parameters. The recorded data is then being processed by an offline algorithm to obtain useful information on the location of the point-source. Our design is cheap to implement; with a total cost of less than one hundred dollars, we are able to detect the general direction of the sound source. The underlying theory and computing algorithm are intuitive and could be used for educational purposes in the future.

II. THEORY AND ALGORITHM

Generally, locating the source of a sound signal using the locations of receivers and the relative time differences of received signals between receivers requires two steps:

- Extract signals to obtain signal arrival times (time shifts);
- Locate the source using the extracted information.

A. Extract Signals and Times

Before locating the sound source, we first need to identify from one of the recordings $r_0(t)$ a signal $s_0(t) = r_0(t+\tau_0), t \in [0,T]$, where τ_0 is the arrival time of the signal and T is the duration of the signal. Then, we need to find the signals $s_i(t)$ in the remaining recordings $r_i(t)$ to get the remaining arrival times. The identification procedure is simple. One can identify a piece of sound signal either by looking at the waveform, listening to the recording, or setting an amplitude threshold. After the signal $s_0(t)$ is identified, it can be compared against $r_i(t)$ to find $s_i(t)$.

Two possible approaches can be used to find $s_i(t)$—the L_2 metric and the auto-correlation.

* Co-first authors in alphabetical order.

1. L_2 Metric

This metric is simply the L_2-norm of the difference (Euclidean distance) [9] between two signals, defined as

$$d(\tau) = \sqrt{\int_0^T \left[\frac{s_0(t)}{Z_0} - \frac{r_i(t+\tau)}{Z_i(\tau)}\right]^2 \mathrm{d}t}, \qquad (1)$$

where τ is the time shift in $r_i(t)$, T is the signal length, and Z_0 and $Z_i(\tau)$ are the normalization factors, defined as

$$Z_0 = \sqrt{\int_0^T s_0^2(t)\mathrm{d}t}; \qquad Z_i(\tau) = \sqrt{\int_0^T r_i^2(t+\tau)\mathrm{d}t}. \qquad (2)$$

The time shift of the recording $r_i(t)$ (arrival time) can be obtained by minimizing $d(\tau)$ and defined as $\tau_i = \arg\min d(\tau)$, from which we extract the arrived signals as $s_i(t) = r_i(t+\tau_i), t \in [0, T]$.

2. Auto-Correlation

The auto-correlation is an inner product of two signals [10] and it has two variants—a normalized version, and an unnormalized version, which is defined as

$$a(\tau) = \frac{1}{Z_0 Z_i(\tau)} \int_0^T s_0(t) \cdot r_i(t+\tau)\mathrm{d}t \qquad (3)$$

In the normalized version, the normalization factors are the same as the ones in Eq. 2; in the unnormalized version, $Z_0 = Z_i = 1$. Then, instead of minimizing $d(\tau)$, one maximizes the correlation $a(\tau)$ to find $\tau_i = \arg\max a(\tau)$ and $s_i(t) = r_i(t+\tau_i), t \in [0, T]$.

One can prove that the normalized version of the auto-correlation is equivalent to the L_2 metric (see Appendix A) and in fact, even the unnormalized version empirically produce the same result as the L_2 metric. Therefore, using L_2 metrics is enough for our work.

B. Locate the Source

After extracting the signals $s_i(t)$ and arrival times τ_i, we can compute the location of the source. In a perfect world where no noise or error exists, the location of the source satisfies

$$|\boldsymbol{x} - \boldsymbol{a}_i| - |\boldsymbol{x} - \boldsymbol{a}_j| = c\,(\tau_i - \tau_j)\,, \forall i, j, \qquad (4)$$

where $\boldsymbol{x}$ is the location of the source, $\boldsymbol{a}_i$ is the location of each recording device, and c is the speed of sound. The equation can be converted into a root-finding problem as

$$|\boldsymbol{x} - \boldsymbol{a}_i| - |\boldsymbol{x} - \boldsymbol{a}_j| - c\,(\tau_i - \tau_j) = 0. \qquad (5)$$

For each pair of i, j, the solution is a branch of hyperbola, and when combined, the solution is the intersection of all hyperbolas. In reality, however, the errors in $\boldsymbol{a}_i$ and τ_i can shift individual hyperbolas in different directions and leave Eq. 5 with no solution. To resolve this issue, we can convert the root-finding problem into a minimization problem by defining a cost function

$$\mathcal{C}(\boldsymbol{x}) = \sum_{i,j} \left[|\boldsymbol{x} - \boldsymbol{a}_i| - |\boldsymbol{x} - \boldsymbol{a}_j| - c\,(\tau_i - \tau_j)\,\right]^2 \qquad (6)$$

and look for $\boldsymbol{x}$ that minimizes the cost function. When there is no error, Eq. 6 gives exactly the same solution as Eq. 2. When there is error, one can show that the $\boldsymbol{x}$ that minimizes the expectation of $\mathcal{C}$ with error is the same as the $\boldsymbol{x}$ that minimizes $\mathcal{C}$ without error (see Appendix B).

III. EXPERIMENTAL PROCEDURE

We test our algorithm in two different setups. In both setups, we choose our testing site as open ground with no trees or other people to avoid unwanted external noise and reflections.

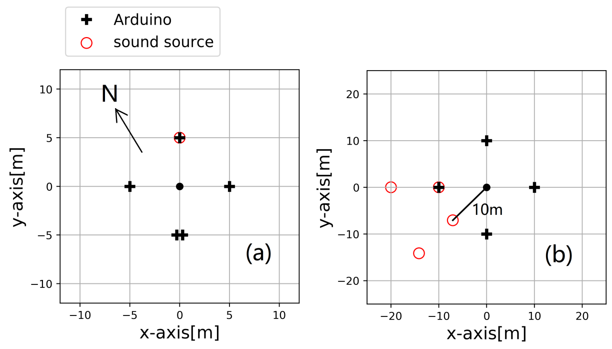


FIG. 1. Experimental setup. The locations of the Arduinos form a square diamond with each vertex 5 or 10 meters from the center. The sound sources (car beep or chirp) are located at the red circle marks.

In the first setup (Fig. 1 (a)), the locations of the Arduinos form a square diamond with each vertex 5 or 10 meters from the center. (We only show a 5-meter diagram in the figure, the 10-meter setup is just a scaled-up version of the 5-meter setup). In this setup, two Arduinos are put at the bottom vertex and the other three Arduinos are put at the other vertices. A car is directly at the top vertex and beeps for roughly 0.5 s. The beep is then recorded by each Arduino. We then shuffle the locations of different Arduinos and repeat the test.

In the second setup (Fig. 1 (b)), the locations of the Arduinos still from a square diamond with each vertex 10 meters from the center. This time, one of the Arduino malfunctions, so only four Arduinos are used. In each test, the car is placed at one of the red circle and plays a chirp. Chirp signal is a special signal whose frequency increases by time with fixed amplitude. In Fig. 2, we show an example of the chirp signal. Since the frequency changes during the signal, it is easier for the algorithm to determine the best arrival times.

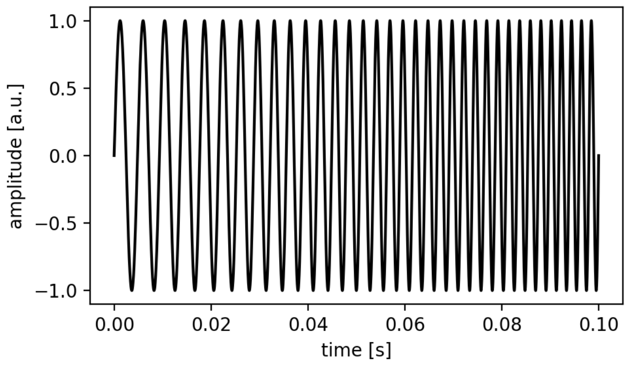


FIG. 2. Example of chirp signal. The unit a.u. means arbitrary unit. The frequency increases linearly from 200Hz to 500Hz.

We show an illustration of the actual test setup in Fig. 3 below.

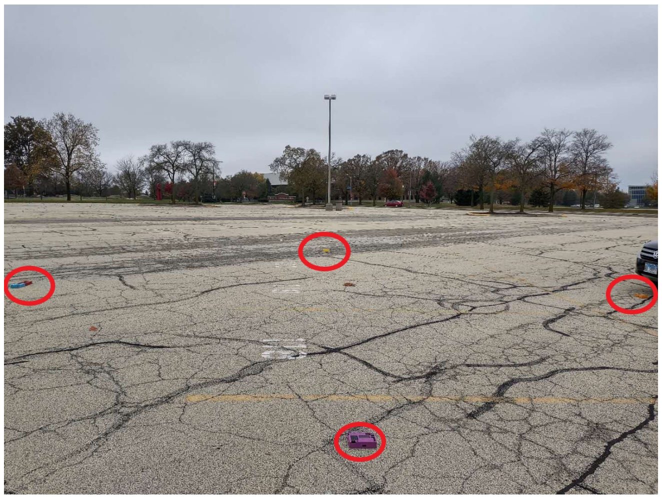


FIG. 3. Illustration of a test setup for Fig. 1 (a). Five Arduinos are put on the vertices of a diamond/square, marked in red circles. There are two Arduinos at the left vertex and the remaining three Arduinos are at the other three vertices. The car is directly at the right vertex and plays a beep sound. In our analysis, we rotate the diamond so that the car is at the top corner, but all relative positions will remain unchanged.

IV. ANALYSIS AND RESULTS

As described in the experimental procedure section, we test our algorithm in two setups. In each test, we collect data and identify the sound signals from one of the recordings. Then we obtain the arrival times (time shifts) in other recordings by comparing them with the signal using the L_2 metric. With all the arrival times, the source location can be calculated by minimizing the cost function. We show below in Fig. 4 and Fig. 5 the recordings, calculated L_2 metric values for different time shifts, and shifted signals for two tests from the two setups.

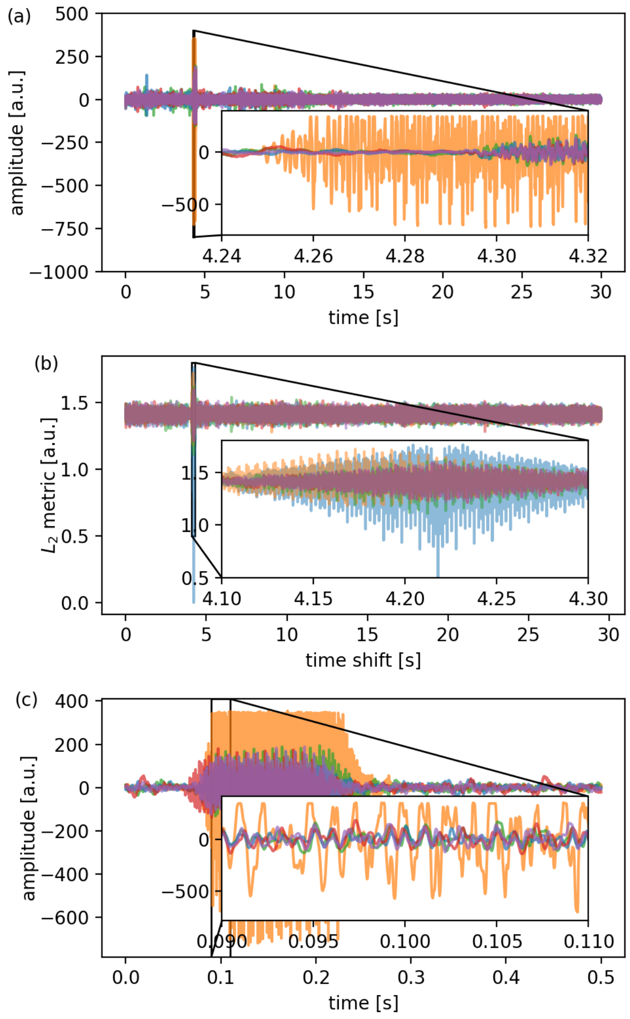


FIG. 4. Example of audio beep recordings in the first setup. The unit a.u. means arbitrary unit. Different colors represent recordings from different microphones. (a) Raw audio recordings $r_i(t)$ from microphones. The zoom-in window shows the start of the audio signals. (b) Computed L_2 metric $d_i(\tau)$ when taking the blue signal as the reference $s_0(t)$ and comparing with respect to all recordings. The zoom-in window shows where the lowest $d_i(\tau)$ for each recording occurs which indicates the optimal time shift τ_i for each recording. (c) The audio signals $s_i(t) = r_i(t+\tau_i)$ extracted from recordings. The zoom-in window shows the matching between signals from different microphones. The orange recording is from the microphone directly at the sound source, which saturated during the beep.

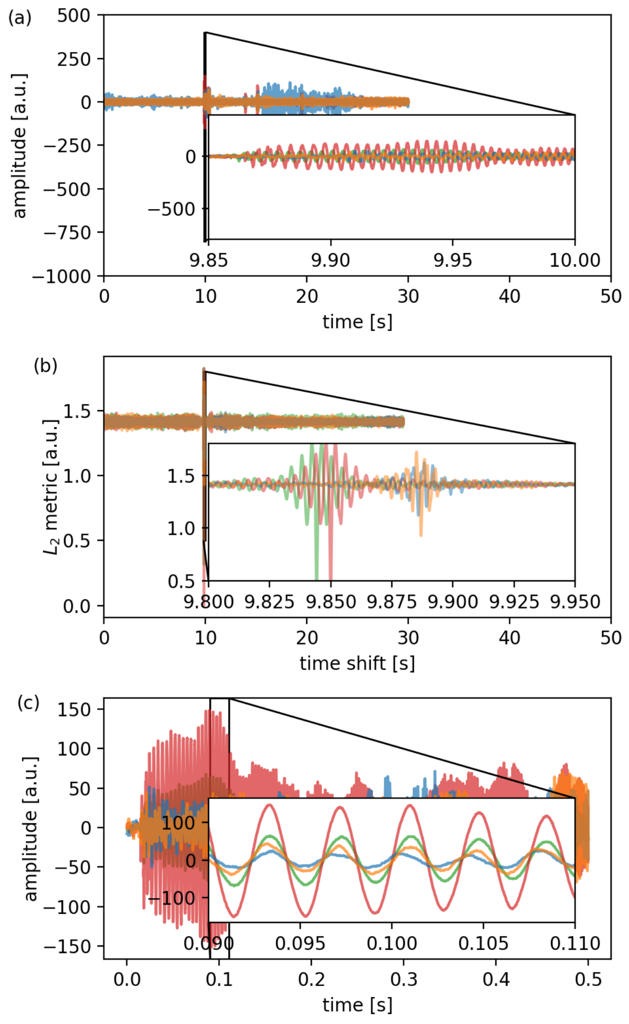


FIG. 5. Example of audio chirp recordings in the second setup. The unit a.u. means arbitrary unit. Different colors represent recordings from different microphones. (a) Raw audio recordings $r_i(t)$ from microphones. The zoom-in window shows the start of the audio signals. (b) Computed L_2 metric $d_i(\tau)$ when taking the red signal as the reference $s_0(t)$ and comparing with respect to all recordings. The zoom-in window shows where the lowest $d_i(\tau)$ for each recording occurs which indicates the optimal time shift τ_i for each recording. (c) The audio signals $s_i(t) = r_i(t + \tau_i)$ extracted from recordings. The zoom-in window shows the matching between signals from different microphones. The microphone at the sound source was covered by a piece of paper, therefore not saturating for this test.

In the first setup, the sound source is a car beep. Since the beep is loud, the microphone directly at the source saturates (orange curve in Fig. 4), so some information from the recording is lost. If the reference signal is taken from the saturated recording, when other recordings are compared against the reference signal, the calculated time shifts τ_i can be incorrect. Therefore we analyze the data from the first setup in two ways.

- Take the reference signal from the saturated recording and hope the saturation is not an issue.
- Take the reference signal from another recording (in our analysis we used the recording at the right vertex).

In the second setup, we place a piece of paper on top of the microphone that is closest to the car, so there is no saturation problem.

As shown in Fig. 4 and Fig. 5, (a) the chirp signals are clearer from noise than beep signals, (b) resulting in a better resolution in L_2 metric to calculate the required time shifts (arrival times), and (c) obtaining better match for the extracted signals. Therefore, we expect the results for the second setup to be better than the first setup.

The results for the first setup are shown in Fig. 6. The blue predictions are calculated by taking the saturated recording as the reference signal, whereas the orange predictions are calculated by taking the recording at the right vertex as the reference signal. As shown in the figure, direction-wise, the orange prediction performs consistently better than the blue prediction. The direction of the orange prediction is very close to the actual direction in all tests. Distance-wise, the two predictions perform differently in different tests.

For Test 3 in Fig. 6, we analyze the effect of multiple recording devices, as shown in Fig. 7. The top figure shows the result with all the recording devices and the bottom four figures show the result with different combinations of three devices. As shown in the figure, the prediction using all devices is in general better than the prediction using only a subset of the devices. This behavior is expected, since the more devices, the closer the cost function we are minimizing is to the expectation of the cost function.

The results for the second setup are shown in Fig. 8. We use the signal from the microphone at the left vertex as the reference signal. Similar to the previous tests, all the direction predictions are in general correct, but the distance predictions are not as good.

For Test 3 in Fig. 8, we also analyze the effect of multiple recording devices. As shown in Fig. 9, the prediction using all devices performs better than the prediction using any subset of them.

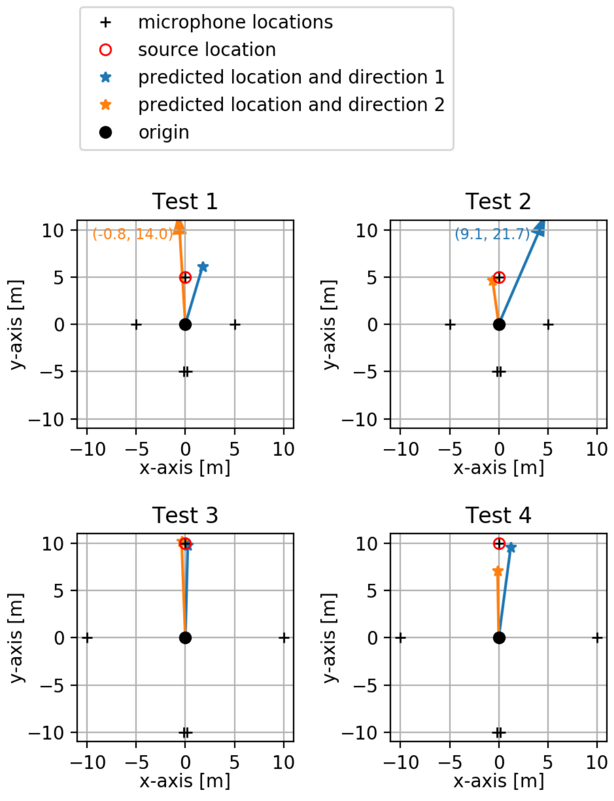


FIG. 6. Four different tests for the direction-finding algorithm. In all tests, the locations of five microphones form a square diamond with two microphones both at the bottom vertex and the other three occupying the other three vertices. The sound source (car beep) is at the red circle at the top corner. The blue prediction is calculated by taking the signal at the top vertex (directly at the sound source, $(0, 5)$ or $(0, 10)$) as the reference $s_0(t)$ (which may saturate and lose information) and comparing with respect to all recordings to extract $s_i(t)$, whereas the orange prediction is calculated by taking the signal at the right vertex $(5, 0)$ or $(10, 0)$ as the reference $s_0(t)$ (which does not saturate but has a smaller amplitude) and comparing with respect to all recordings to extract $s_i(t)$.

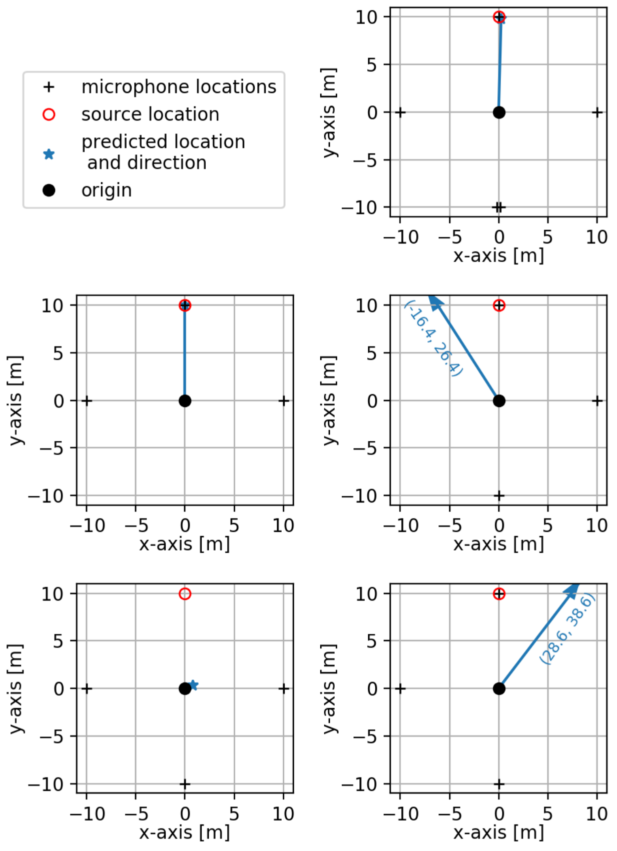


FIG. 7. Results for different combinations of detectors for Test 3 in Fig. 6. The top one uses all detectors whereas the bottom four use different combinations of three detectors (black crosses). The prediction is calculated by taking the signal at the top vertex as the reference $s_0(t)$ and comparing with respect to all recordings to extract $s_i(t)$.

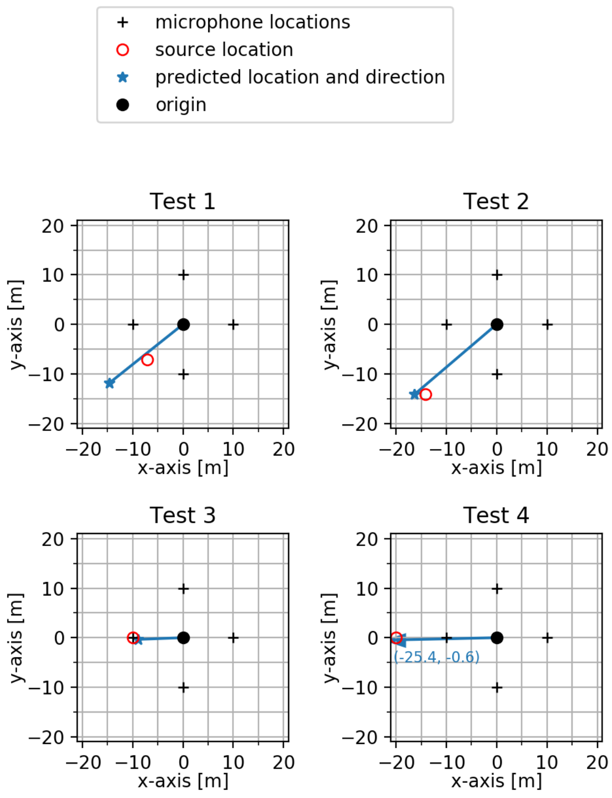


FIG. 8. Four different tests for the direction-finding algorithm. In all tests, there are four microphones whose locations form a square diamond. The sound source (chirp) is at the red circle at different locations for different tests. The prediction is calculated by taking the signal at the left vertex $(-10, 0)$ as the reference $s_0(t)$ and comparing with respect to all recordings to extract $s_i(t)$.

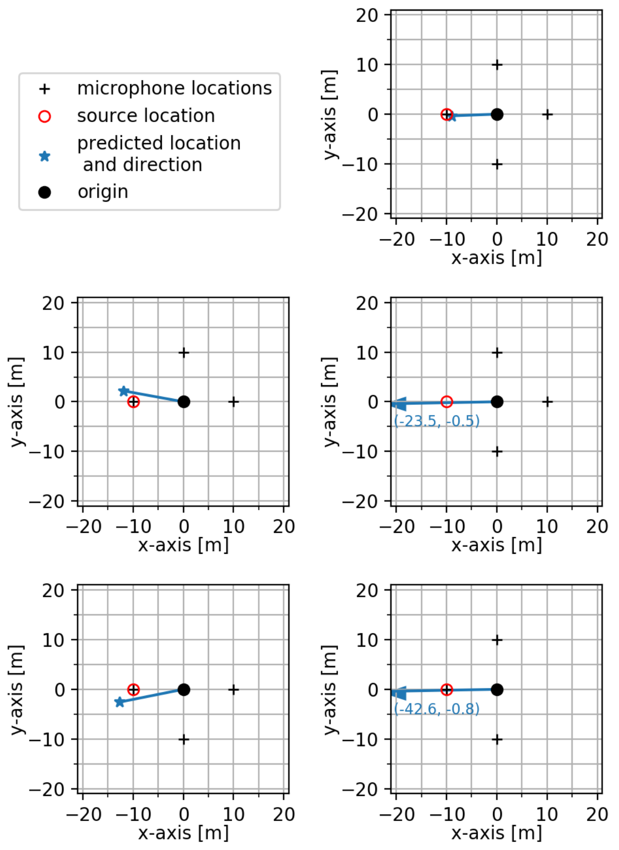


FIG. 9. Results for different combinations of detectors for Test 3 in Fig. 8. The top one uses all detectors whereas the bottom four use different combinations of three detectors (black crosses). The prediction is calculated by taking the signal at the left vertex as the reference $s_0(t)$ and comparing with respect to all recordings to extract $s_i(t)$.

V. DISCUSSION

In general, the algorithm performs reasonably well in all tests. The predicted direction (if the recorded audio does not saturate) always points close enough to the actual source. The predicted distance, on the other hand, is less accurate. We believe that predicting distance is intrinsically harder than predicting directions, and since the objective of this project is acoustic direction finding, we claim the project a success.

Nevertheless, many improvements can be made. Currently, we only perform tests in a well-controlled environment where the noise is small and the signal is clear. Limited by the signal-to-noise ratio of our recording devices, a real-world test is not plausible at this moment, but with better recording devices, we should be able to test our algorithm in a more complicated environment. In addition, given the stochastic nature of errors, by adding additional recording devices, we should expect the errors being averaged out, improving the accuracy both direction-wise and distance-wise.

VI. CONCLUSION

We show a robust algorithm to locate a sound signal using simple recording devices. The algorithm averages out the error from individual recording devices by minimizing a cost function. Our algorithm generates accurate direction predictions despite the poor signal-to-noise ratio of our recording devices. The distance predictions can be improved by adding additional recording devices or switching to more advanced recording devices. We expect the algorithm to have great potential in various applications such as locating crimes or car accidents.

VII. ACKNOWLEDGEMENT

Some of the codes to record audios and process audio files are adapted from Professor George D. Gollin's repository. We thank Professor George D. Gollin and Shubhang Goswami for their insightful discussions and generous support on this project.

Appendix A: L_2 Metric vs Normalized Auto-correlation

We claim that the L_2 metric (Eq. 1) and normalized auto-correlation (Eq. 3) are equivalent. Here we give a proof. We expand the square of Eq. 1.

$$\begin{aligned} d^2(\tau) &= \int_0^T \frac{s_0^2(t)}{Z_0^2} + \frac{r_i^2(t+\tau)}{Z_i^2(\tau)} - 2\frac{s_0(t)}{Z_0}\frac{r_i(t+\tau)}{Z_i(\tau)}\mathrm{d}t \\ &= \frac{1}{Z_0^2}\int_0^T s_0^2(t)\mathrm{d}t + \frac{1}{Z_i^2(\tau)}\int_0^T r_i^2(t+\tau)\mathrm{d}t \\ &\quad - \frac{2}{Z_0 Z_i(\tau)}\int_0^T s_0(t)\cdot r_i(t+\tau)\mathrm{d}t \\ &= 2\left[1-a(\tau)\right], \end{aligned} \tag{A1}$$

where $a(\tau)$ is the normalized auto-correlation defined in Eq. 3. Therefore, minimizing the L_2 metric is equivalent to maximizing the normalized auto-correlation.

Appendix B: Minimize Cost Function with Error

In Eq. 6, we define a cost function and claim that when errors exist, the cost function is still valid. Here we provide a proof. Since the spatial error of the microphones $\Delta \boldsymbol{a}_i$ depends on the tape measure, which we can guarantee is less than 1 cm. The temporal error $\Delta\tau_i$ of the arrival signals, however, is much less accurate. Even though Arduinos can provide a time accuracy of $\sim 1\,\mu$s, the L_2 metric cannot extract signals at such an accuracy. Depending on the background noise and the shape of the signal, the error from the L_2 metric can range from ~ 1 ms to as large as ~ 10 ms. Therefore, it is safe to assume that $c\Delta\tau_i \gg |\boldsymbol{a}_i|$, where c is the speed of sound. Thus, let's add the error term into Eq. 6.

$$\begin{aligned} \mathcal{C}_{\text{with error}}(\boldsymbol{x}) = \sum_{i,j}\Big[&|\boldsymbol{x}-\boldsymbol{a}_i| - |\boldsymbol{x}-\boldsymbol{a}_j| - c\,(\tau_i-\tau_j) \\ &-c\,(\Delta\tau_i - \Delta\tau_j)\Big]^2 \\ = \sum_{i,j}\Big[&|\boldsymbol{x}-\boldsymbol{a}_i| - |\boldsymbol{x}-\boldsymbol{a}_j| - c\,(\tau_i-\tau_j)\Big]^2 \\ &-2\Big[|\boldsymbol{x}-\boldsymbol{a}_i| - |\boldsymbol{x}-\boldsymbol{a}_j| - c\,(\tau_i-\tau_j)\Big] \\ &\cdot c\,(\Delta\tau_i - \Delta\tau_j) \\ &+\Big[c\,(\Delta\tau_i - \Delta\tau_j)\Big]^2. \end{aligned} \tag{B1}$$

Now, let's look at the expectation value of $\mathcal{C}_{\text{with error}}$

$$\begin{aligned} \langle\mathcal{C}_{\text{with error}}(\boldsymbol{x})\rangle &= \sum_{i,j}\Big[|\boldsymbol{x}-\boldsymbol{a}_i| - |\boldsymbol{x}-\boldsymbol{a}_j| - c\,(\tau_i-\tau_j)\Big]^2 \\ &\quad -2\Big[|\boldsymbol{x}-\boldsymbol{a}_i| - |\boldsymbol{x}-\boldsymbol{a}_j| - c\,(\tau_i-\tau_j)\Big] \\ &\quad \cdot c\langle\Delta\tau_i - \Delta\tau_j\rangle \\ &\quad +c^2\langle(\Delta\tau_i - \Delta\tau_j)^2\rangle \\ &= \sum_{i,j}\Big[|\boldsymbol{x}-\boldsymbol{a}_i| - |\boldsymbol{x}-\boldsymbol{a}_j| - c\,(\tau_i-\tau_j)\Big]^2 \\ &\quad +c^2\Big[\mathrm{Var}(\tau_i) + \mathrm{Var}(\tau_i)\Big] \\ &= \mathcal{C}(x) + \text{constant}. \end{aligned} \tag{B2}$$

Therefore, the x that minimizes the expectation of $\mathcal{C}_{\text{with error}}(\boldsymbol{x})$ is the same as the x that minimizes $\mathcal{C}(x)$.

Appendix C: Data Acquisition Device

Our Arduino-based data acquisition devices could record 10-bit 40-kHz audio files with time drift error smaller than 10 ppm (part per million). The device's main components include an Arduino Mega2560, a MicroSD card breakout board, a microphone and associated amplifier, a GPS breakout board, and a BME680 gas sensor [11]. We think the data acquisition devices are functioning well and the recording quality is quit good given the elementary AVR processors. In the future, we may upgrade the micro-controllers to more advanced ones to improve recording quality. We could also upgrade the microphones or install parabolic reflectors to obtain better sound signals.

With a third-party library (SdFat), the Arduino could serve as an analog-digital converter (ADC) of at most 10-bit 40 kHz or 8-bit 55 kHz [12]. The Arduino measures the microphone output's analog voltage and stores it to a MicroSD card as a binary file using the MicroSD breakout board. The GPS breakout board provides a highly accurate time signal (PPS, or Pulse-Per-Second), with which the time drift is calibrated. The BME680 collects the temperature, humidity, and air pressure information to estimate the speed of sound [13].

We use the Adafruit electret microphone amplifier board (MAX4466 with adjustable gain) [14] to collect audio. The board comes with a 20Hz-20kHz electret microphone and an operational amplifier. According to the manufacturer, the amplifier has excellent power supply noise rejection and sounds quieter than other similar microphone amplifier breakout boards on the market. It is suitable for projects such as voice changers, audio recording and sampling, and audio-reactive projects that use Fast Fourier Transform . [14] The microphone board's OUT pin is connected to the Arduino analog pin directly, and no extra audio amplifiers are needed.

1. Interrupt-driven analog bin logger

The Arduino Mega 2560 provides a default function, `AnalogRead()`, to perform analog voltage measurement. However, this function costs more than 100 microseconds to perform a test, [15] which is too slow for our usage. To speed up the measurement, some modifications need to be made to the origin process. The library SdFat helped us implement the modifications, and we would build our software basing on it.

The first is to change the prescaler of the ADC clock. The processor frequency is 16 MHz, and the ADC runs slower than the central processor by a prescaler index. The default prescaler is 7, meaning that the ADC runs at $16/2^7 = 0.125$ MHz or 125 kHz. It would cost the ADC 13 cycles to finish a measurement, so the time needed to conduct a measurement is about $1/125$ kHz $\times\ 13 = 104\ \mu$s. If we change the Prescaler to 4, the time for a single measurement is reduced to $1/16$ MHz $\times\ 2^4 \times 13 = 13\ \mu$s. This change would increase the ADC's speed without sacrificing much of the accuracy because it is shown by both the datasheet and the earlier experiments that the ADC accuracy would not drop significantly for an ADC clock up to 1 MHz. [15]

The second technique is to use buffers in SRAM (static random access memory of the Arduino, 8k for Arduino Mega2560) [16] for the collected data. The transformation of data from the Arduino to the MicroSD card is done via the SPI interface. Generally, the maximum bandwidth of SPI allows us to record 10-bit 40 kHz audios. However, we can not transfer the data byte by byte because this practice would repeatedly start and close the MicroSD breakout board, which we believe would take much more time than transmitting the data altogether. Besides, the MicroSD card sometimes pauses all the interactions and cleans its internal cache, which may take up to hundreds of microseconds [17]. Therefore, we have to store the collected data to the Arduino when the MicroSD card is not responding. The solution is to use the extra space in the SRAM as the buffer to store the data temporarily and send them to the MicroSD card later. We can learn how much RAM space is left from the Arduino IDE terminal after uploading the DAQ program. We have to ensure that we did not use up all the remaining space in SRAM because other local variables in the program need SRAM space too. If that happens, there would be some SRAM conflicts, and they may cause the Arduino to crash.

The third is to use the so-called "Interrupt-driven ADC" technique [18]. The ATMega2560, the microcontroller the Arduino Mega2560 runs on, can start an ADC and then provide an interrupt `ADC_vect` when it is done converting. It means that while the ADC is running, the central processor can be executing other code. We would be moving the data to the MicroSD card from SRAM buffers while the ADC is running and collecting data. We would also use a `timer1` interruption to ensure that the ADC runs at our appointed frequency.

In conclusion, our data acquisition device software runs in this way: we have a queue of buffers in the SRAM of the Arduino, and the MicroSD card is ready to receive data. After we start recording, the Arduino ADC begins to do the analog measurement and store the data in the SRAM buffers, at the same time, the `timer1` interruption ensures that every measurement is completed during the specified time interval (in our case, every $1/40$ kHz $= 25\ \mu$s). The measurement is conducted in an interrupt service routine function, meaning that we can do something else (moving data to MicroSD card) while performing the measurement. Every time one buffer is filled, the ADC would start to store data in the next buffer in the queue, and the Arduino would begin to move the data to the MicroSD card and clean the filled buffer for future usage.

2. Time synchronization and calibration

To localize the sound source, we need at least 4 data acquisition devices at different locations and measure the delay time of the arriving signals between different devices. Thus, the problem is transferred to a multilateration problem, which can be solved by geometric reconstruction.

However, to ensure that we can obtain accurate time of arrival (TOA) information, we have to synchronize the clocks of different devices and keep their speeds correct. In other words, we have to eliminate or minimize the time drift of the devices.

Suppose the device recorded a series of data a_i, with the index $i = 0, 1, 2, \ldots$. The exact time when a_i is recorded can be written as

$$t_i = t_0 + \alpha \Delta t \times i \tag{C1}$$

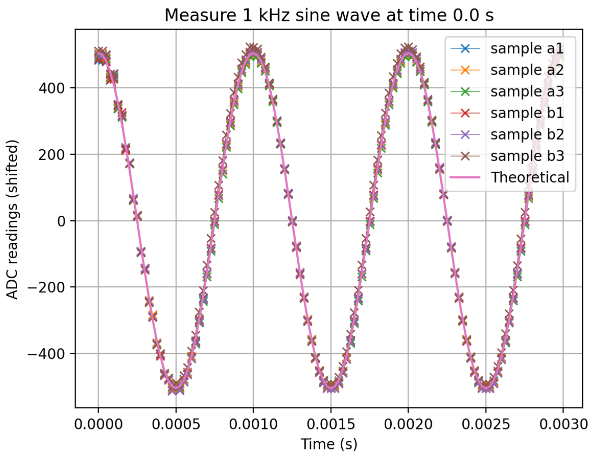
where t_0 is the start time of recording, Δt is the nominal time interval between two data points (25 μs for a sample rate of 40000 kHz), and α is the time calibration coefficient such that $\alpha\Delta t$ becomes the real interval between two data points.

The PPS (Pulse-Per-Second) pin of the GPS provides a 100 ms pulse every second with time drift less than 100 ns. [19] So we would use the rising edge of the PPS pulse to synchronize the beginning of recording on different devices. It is done by an interrupt service routine that starts the recording exactly when the rising edge arrives. Therefore, the beginning time t_0 of different devices would be the same. By later tests (Fig. 10(a) and Fig. 10(b)), the errors of the starting points are small enough to be neglected compared with other errors.

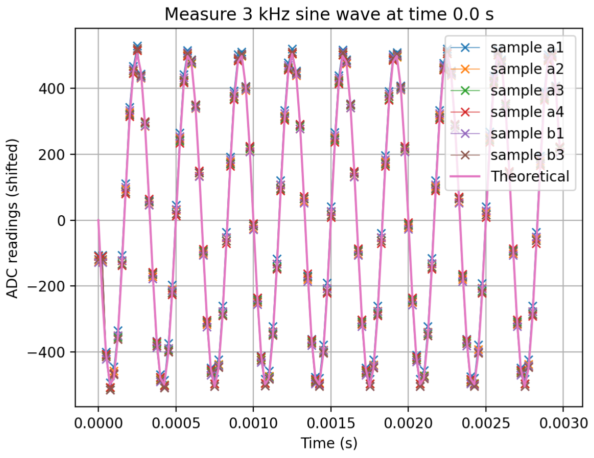
The internal clocks (crystal oscillator) of different Arduino also tick at different speeds. The differences are caused by the different qualities of the crystal oscillators. The temperature also plays a role in the speed. According to our measurement, when the environment temperature increases 1 °C, the clock would be slower by about 13 μs per second. The time calibration coefficient is measured by comparing the time interval between two PPS pulses and the Arduino internal clock's time interval. The ratio between the time difference by the internal clock and the number of PPS pulses is the time calibration coefficient. The time calibration coefficient is measured both before and after recording and their average is taken to reduce error.

Hence, we know the t_0 and α of different devices, thus knowing each data point's exact time. Following, we can use a resample function [20] to adjust the sample rate of the sample so that the data's comparison could be more convenient.

Also, the total number of buffers we use controls when the Arduino stops recording. Each buffer holds 254 words, and one word corresponds to 25-μs 10-bit sample. Therefore, one buffer corresponds to a recording of 6.35 ms. We can calculate how many buffers we need to record a sample of the given length, and we would stop the recording when all buffers are used.



(a)Calibrated audio recording of 1 kHz sine wave at $t = 0$ s.



(b)Calibrated audio recording of 3 kHz sine wave at $t = 0$ s.

FIG. 10. Sign wave of 1 kHz and 3 kHz at $t = 0$ s with calibration. Data labeled by "sample a" are collected by Breadboard 1 and Data labeled by "sample b" are collected by Breadboard 2.

Appendix D: Accuracy Analysis

To verify that the data collected by our 10-bit 40-kHz DAQ is reliable, we use commercial devices in Fig. 11 to test them, including a function generator (Tektronix AFG1022, time accuracy ± 1 ppm) and an oscilloscope (RIGOL DS1054). We generated some sine waves of specific frequencies and let the DAQ measure these signals. Following, we used a computer to do some offline data processing and calculated the frequencies of the data collected by the DAQ. By comparing the differences in the frequencies, we could know how accurate are our data acquisition devices.

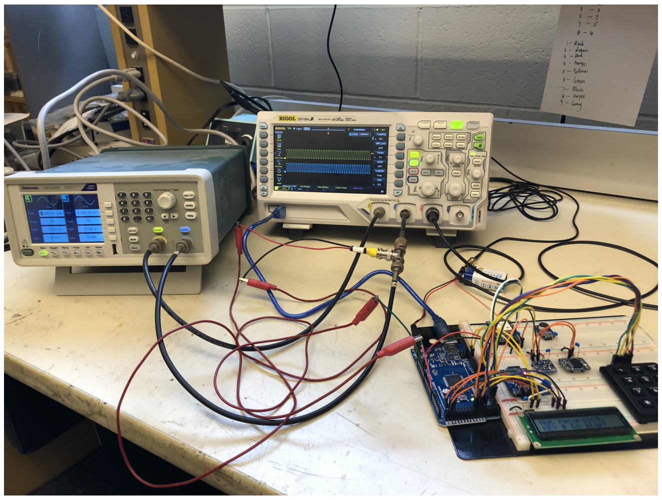


FIG. 11. The oscilloscope (RIGOL DS1054) and the function generator (Tektronix AFG1022). The breadboard version of our DAQ is also in the picture.

First, we measured the time calibration coefficients of all the devices we used. These coefficients are related to the speeds of the internal clocks of the Arduinos. The time drifts caused by this reason can vary from hundreds to thousands of PPM. The average calibration coefficients α of all our devices at the sample temperature are listed in Table I.

Device	Ave. α	Drifts
Blue	$1 + 1.0 \times 10^{-3}$	1000 ppm
Orange	$1 + 1.8 \times 10^{-3}$	1800 ppm
Purple	$1 + 2.8 \times 10^{-4}$	280 ppm
Red	$1 + 1.0 \times 10^{-3}$	1000 ppm
Yellow	$1 + 1.2 \times 10^{-3}$	1200 ppm
Breadboard 1	$1 - 2.1 \times 10^{-3}$	-2100 ppm
Breadboard 2	$1 + 7.0 \times 10^{-5}$	70 ppm

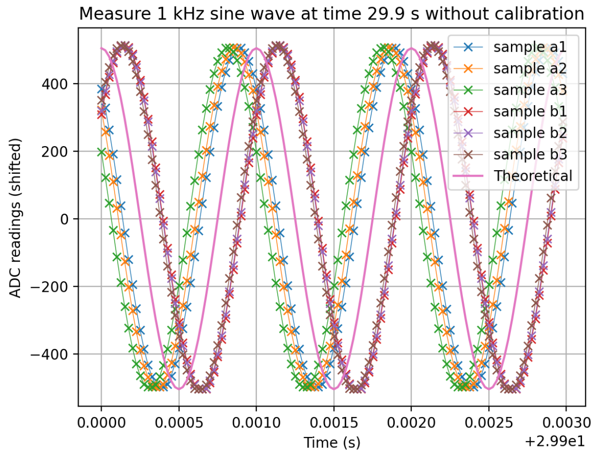
TABLE I. Average time calibration coefficients α of all the devices we used, and their drifts before calibration.

After we applied the time calibration method mentioned in Appendix C, the drifts of Breadboard 1 and Breadboard 2 are significantly reduced. Comparing the frequencies by the function generator and our DAQ, we found the final time drifts listed in Table II. We believe that these drifts come from the time calibration coefficients. First, to achieve the time calibration coefficients, we used the `micros()` function, whose time resolution is 4 μs instead of 1 μs, resulting in some potential errors. Second, there may be some temperature fluctuation during the measurement, disturbing the internal clocks' speed and causing some non-linear time drifts.

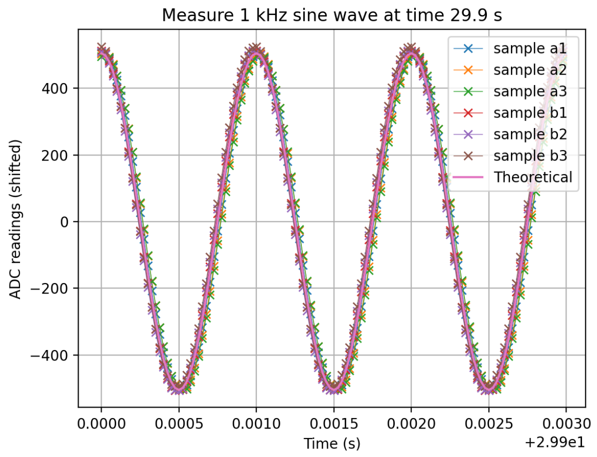
We compared the calibrated data and uncalibrated data. The data are calibrated using the information in Table I. In Fig. 12(a) and Fig. 10(b), we measured the 1-kHz sign wave at $t = 29.9\,\mathrm{s}$ without and with calibration. In Fig. 16(a) and Fig. 16(b), we measured the 3-kHz sign wave at $t = 29.9\,\mathrm{s}$ without and with calibration. Data labeled by "sample a" are collected by Breadboard 1, and Data labeled by "sample b" are collected by Breadboard 2. The figures confirm that the data are well synchronized and calibrated.

Device	Ave. drifts
Breadboard 1	-4.5 ppm
Breadboard 2	1.5 ppm

TABLE II. Average time drifts of Breadboard 1 and 2 after time calibration

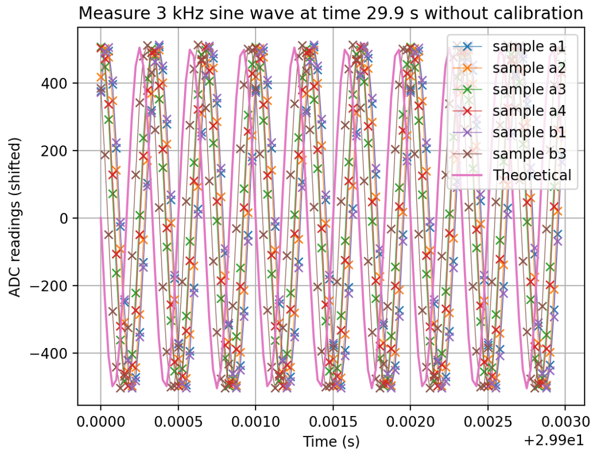


(a)Uncalibrated audio recording of 1kHz sine wave at $t = 29.9\,\mathrm{s}$.

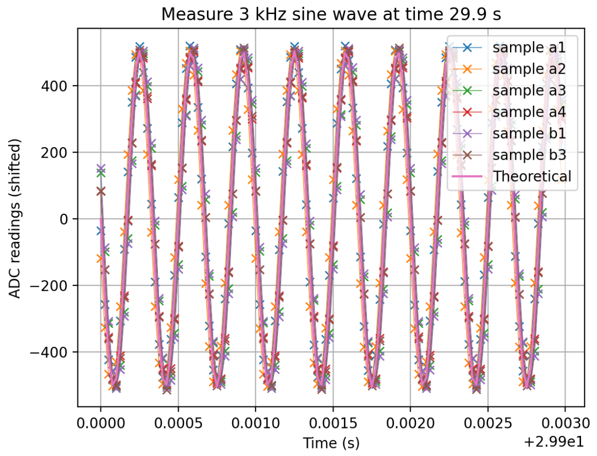


(b)Audio recording of 1kHz sine wave at $t = 29.9\,\mathrm{s}$ calibrated by the GPS PPS signal.

FIG. 12. Sign wave of 1 kHz at $t = 29.9\,\mathrm{s}$ without and with calibration. Data labeled by "sample a" are collected by Breadboard 1 and Data labeled by "sample b" are collected by Breadboard 2.



(a)Uncalibrated audio recording of 3kHz sine wave at $t = 29.9\,\mathrm{s}$.



(b)Audio recording of 3kHz sine wave at $t = 29.9\,\mathrm{s}$ calibrated by the GPS PPS signal.

FIG. 13. Sign wave of 3 kHz at $t = 29.9\,\mathrm{s}$ without and with calibration. Data labeled by "sample a" are collected by Breadboard 1 and Data labeled by "sample b" are collected by Breadboard 2.

Appendix E: Error in Calculated Time Shift

In this section, we show the error in the calculated time shifts τ_i in our tests. In particular, we demonstrate the accuracy in two ways.

- We use the actual distance information of the source and all microphones to compute the theoretical time shift τ_{theory} of each recording, and show the difference between the computed time shifts using the L_2 metric and the theoretical time shifts.
- We estimate the error using the L_2 metric values.

Specifically, for the second method, we first convert the L_2 metric values to normalized auto-correlation values according to Eq. A1 as

$$a(\tau) = 1 - \frac{d^2(\tau)}{2}. \tag{E1}$$

Then, we notice that audio signals generally center at zero ($\mathrm{mean}(s_i) \approx 0$), so the normalized auto-correlation is actually the correlation coefficient.

$$\begin{aligned} a(\tau) &= \frac{\frac{\int_0^T s_0(t)\cdot r_i(t+\tau)\mathrm{d}t}{T}}{\sqrt{\frac{\int_0^T s_0^2(t)\mathrm{d}t}{T}\frac{\int_0^T r_i^2(t+\tau)\mathrm{d}t}{T}}} \\ &= \frac{\mathrm{Cov}[s_0 \cdot r_i]}{\mathrm{Std}[s_0]\cdot\mathrm{Std}[r_i]}(\tau) \\ &= \rho_{s_0 r_i}(\tau). \end{aligned} \tag{E2}$$

The standard error of correlation coefficient [21] is

$$\mathrm{SE}(\rho) = \frac{\sqrt{1-\rho^2}}{\sqrt{N-2}}, \tag{E3}$$

where N is the number of time bins. Then, we estimate the error τ_{err} as $|\tau_2 - \tau_1|$ where τ_1 and τ_2 are the first and last roots of $a(\tau) = \max(a) - \mathrm{SE}(\max(a))$.

Setup	$\lvert\boldsymbol{a}-\boldsymbol{x}\rvert$	$\tau - \tau_{\text{theory}}$	τ_{err}
1 (Beep)	0 m	N/A	N/A
	14.142 m	-5.41×10^{-3} s	9.88×10^{-3} s
	14.142 m	-5.48×10^{-3} s	8.31×10^{-3} s
	20 m	1.282×10^{-2} s	1.175×10^{-2} s
	20 m	8.24×10^{-3} s	9.09×10^{-3} s
2 (Chirp)	0 m	N/A	N/A
	14.142 m	3.20×10^{-4} s	7.50×10^{-4} s
	14.142 m	3.31×10^{-4} s	2.50×10^{-4} s
	20 m	5.57×10^{-4} s	1.250×10^{-3} s

TABLE III. Error in the calculated time shifts. The data from Test 3 in each setup is analyzed. The distance of each microphone to the source is shown in $|\boldsymbol{a}-\boldsymbol{x}|$. The actual errors $\tau - \tau_{\text{theory}}$ are calculated using the actual distance information of the source and microphones. The τ_{err} values are calculated using the L_2 metric values. In each test, the time shifts are compared with the time shifts at 0 m, so the actual error is not available at this location.

The errors for Test 3 in each setup are shown in Table III. The actual error is generally at the same order of magnitude as the τ_{err}. We observe that the error increases as the distance increases, because of the decrease in sound volume as well as the signal-to-noise ratio. In addition, the error when using the chirp signal is much smaller than the error when using the beep signal, which is expected since the frequency of the chirp signal changes with respect to time, allowing for an easier match of different recordings. Besides, we know Arduino real-time clock has much better accuracy than $1 \times 10^{-4}\,\mathrm{s}$, so the total error in time is dominated by the error from L_2 metric, which means our algorithm is bottlenecked by the poor signal-to-noise ratio of the microphones.

Appendix F: Cover Design

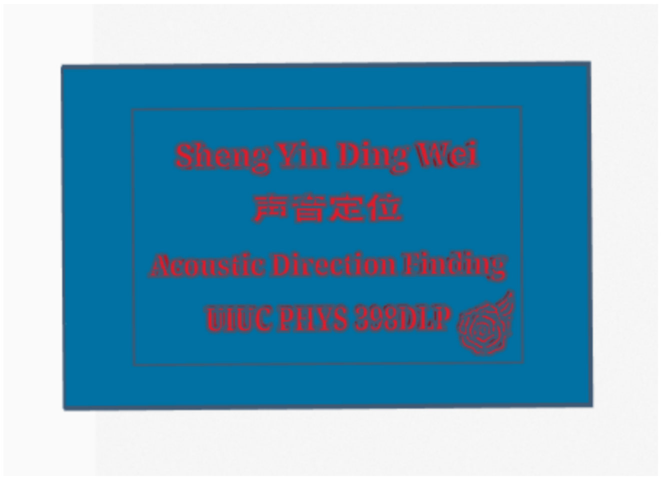


FIG. 14. Case lid design of breadboards. The top row is the pinyin (pronunciations) of the Chinese characters in the second row, which means exactly acoustic direction-finding.

We designed a unique case lid for our breadboards, which contains four lines of words. The first three lines are the same meaning: the first line is the Chinese pinyin (pronunciation), the second line is the Chinese characters, and the third line is the English translation for the Chinese characters.

Appendix G: Recording Devices

We show our Arduino-based recording devices below. The final PCB version of the recording devices (Fig. 15) is used for all the tests. The breadboard version (Fig. 17) is studied to understand the behavior of individual modules and to develop and test the data acquisition software.

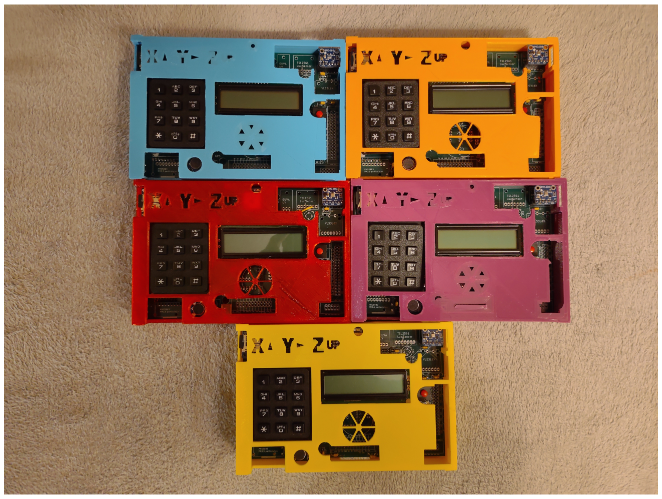
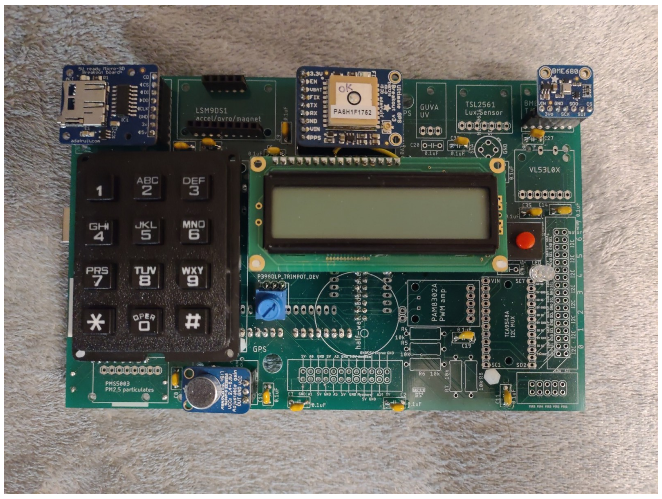
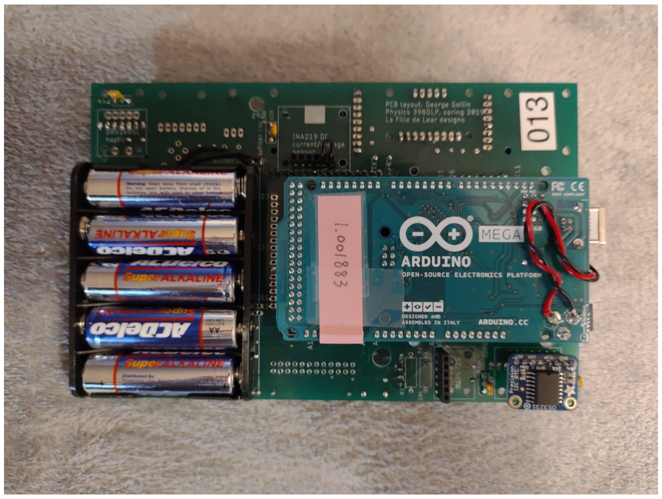


FIG. 15. Five final PCB version of the recording devices.



(a)Front side of PCB version of the recording device.



(b)Back side of PCB version of the recording device.

FIG. 16. Front side and Back side of the PCB version of the recording device.

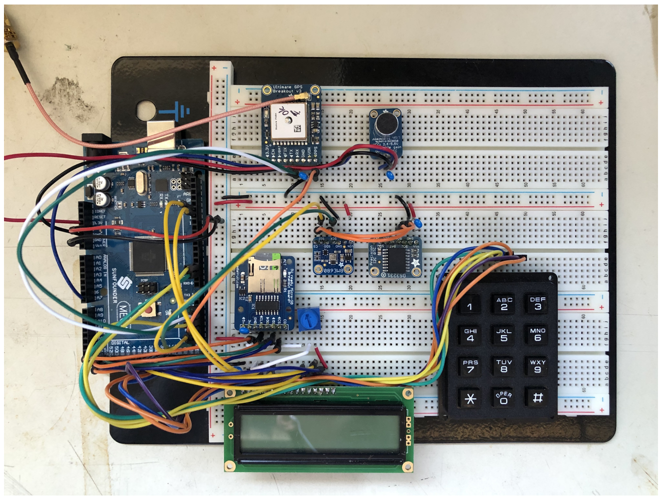


FIG. 17. A breadboard version of the the recording device.

[1] A. Blanco, “Long-range artillery sound ranging: PASS GR-8 sound ranging data,” (1979).

[2] G. Valenzise, L. Gerosa, M. Tagliasacchi, F. Antonacci, and A. Sarti, in *2007 IEEE Conference on Advanced Video and Signal Based Surveillance* (2007) pp. 21–26.

[3] A. Schuijf, in *Hearing and Sound Communication in Fishes*, edited by W. N. Tavolga, A. N. Popper, and R. R. Fay (Springer New York, New York, NY, 1981) pp. 267–310.

[4] H. E. de Bree, W. F. Druyvesteyn, E. Berenschot, and M. Elwenspoek, in *Technical Digest. IEEE International MEMS 99 Conference. Twelfth IEEE International Conference on Micro Electro Mechanical Systems (Cat. No.99CH36291)* (1999) pp. 124–129.

[5] F. Gustafsson and F. Gunnarsson, in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP ’03).*, Vol. 6 (2003) pp. VI–553.

[6] J. P. Dmochowski, J. Benesty, and S. Affes, IEEE Transactions on Audio, Speech, and Language Processing **15**, 2510 (2007).

[7] H. E. de Bree, Acoustic Australia **31**, 91 (2003).

[8] N. La Vigne, P. Thompson, D. Lawrence, and M. Goff (2019).

[9] I. Dokmanic, R. Parhizkar, J. Ranieri, and M. Vetterli, IEEE Signal Processing Magazine **32**, 12 (2015).

[10] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. (McGraw Hill, Boston, 2002).

[11] G. Gollin, “Phys398 daq schematic and pcb layout,” (2020).

[12] B. Greiman, “Sdfat,” `https://github.com/greiman/SdFat` (2020).

[13] “Speed of sound in humid air,” (2020).

[14] Adafruit.com, “Electret microphone amplifier - max4466 with adjustable gain,” (2020).

[15] N. Gammon, “Adc conversion on the arduino (analogread),” (2015).

[16] Arduino.cc, “Foundations: Memory,” (2018).

[17] S. de Wit, “Microsd performance on memory-constrained devices,” (2019).

[18] G. Sweeney, “Interrupt-driven adc with arduino,” (2020).

[19] Adafruit.com, “Adafruit ultimate gps breakout - 66 channel w/10 hz updates - version 3,” (2017).

[20] PyPI.org, “resample 1.0.1,” (2020).

[21] E. Garcia, “A tutorial on standard errors,” (2016).