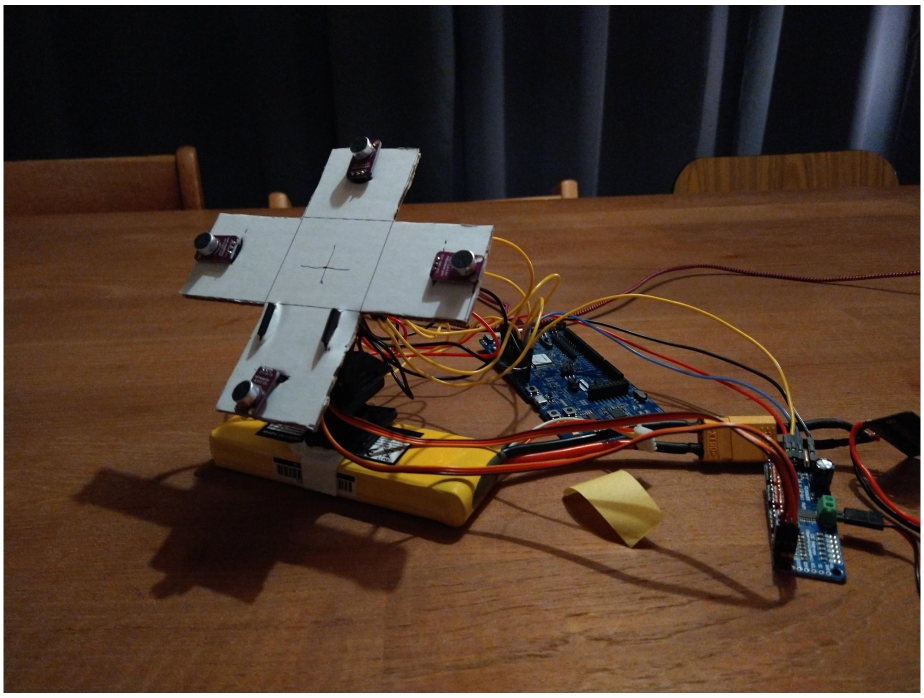
Folley: real-time mosquito noise origin locator

Report of the 5LIU0 DBL project

Henk Oordt
1717510

February 4, 2022



1 Introduction

Project ‘Folley’ is aimed at the design and construction of a real-time sound origin locator. ‘Folley’ uses audio signal analysis to detect and locate in 3D space the origin of the buzzing sound of mosquitoes.

In order to locate the sounds origin, ‘Folley’ samples audio signal from an array of four analog microphones. The sampled signals are then analyzed in order to calculate a Time-Delay-angle-Of-Arrival (TDOA) [2] which, along with the known microphone setup dimensions, can be used to calculate the azimuth and altitude angles of the origin with respect to the microphone array of the device.

In order to develop the TDOA analysis software, a set of Matlab [8] scripts were written, which given the raw audio signal measurements, can calculate the azimuth and altitude angles of the sounds origin with respect to the microphone array. Essentially, in these scripts all of the signal analysis calculations necessary to reach the project goals are implemented. These Matlab scripts will serve as a basis and a means of verification for the Rust implementation of the algorithm in firmware. Upon completion of the Matlab scripts and tweaking of parameters, the calculations have been re-implemented in Rust [9], in order for the analysis to be done by the microcontroller on the nRF52840 DK [4] board in real time. A simple command line application written in Rust that can communicate with the device and that converts raw microphone measurements to Matlab input files was developed as well.

This project focuses solely on the implementation of the TDOA analysis, as well as its evaluation. In this project, a testing environment was set up. This environment consists of a simple firmware application that is able to sample microphone data, and communicate these samples with the command line application that records them. The environment having been set up, a Matlab script has been implemented that is able to do the TDOA analysis based on four sine waves with separate phase differences, but with the same frequencies. Once this Matlab script had finished, the TDOA analysis was re-implemented in firmware, so that it can be done with microphone samples in real time. With this project done, ‘Folley’ should be able to locate origins of predfdined sine wave sounds, coming from a waveform generator, as well as recordings of real mosquito buzzing sound.

The software developed in this project, as well as the data used to verify calculations, are available online within the project repository [3] and in the accompanying archive. This report references paths within the archive and repository.

2 Problem specification

Mosquitos emit a continuous buzzing sound in which certain frequencies and harmonics are present when flying around. This sound can be used to estimate the location of the insect. A device can use a set of microphones to analyze the sound signals as it is recorded by multiple microphones and obtain information about the location of the insect.

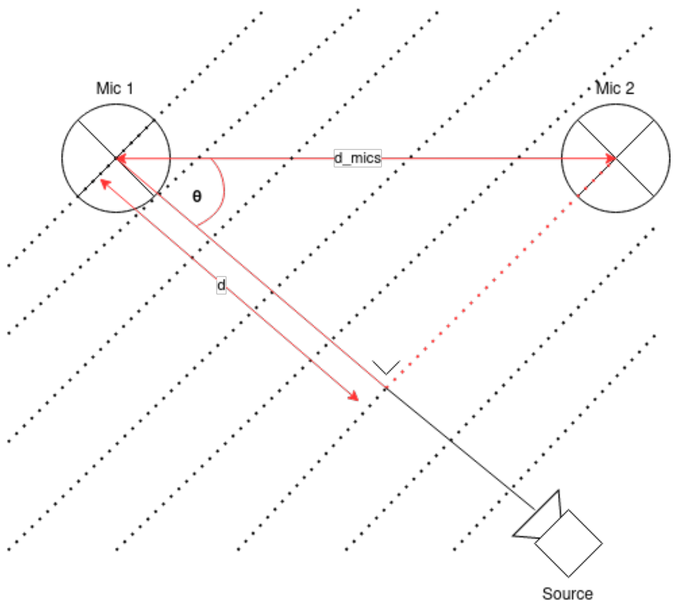


Figure 1: Problem overview

As the insect moves, the sound will come from different angles from the device's perspective. For both the x and y axis, the time of arrival of the buzzing sound can be used to calculate the angle from which it originated. In figure 1, a schematic overview of the problem is given for a single axis. A sound source produces a sound. The origin is assumed to be infinitely far away, so the sound is moving in straight waves when it arrives at the microphone array. In this example, the sound hits microphone 2 before it hits microphone 1. Let d be the distance travelled by the sound between the moment it hits microphone 2 and the moment it hits microphone 1 and d_{mics} be the distance between the microphones. Then, a right triangle can be formed with a hypotenuse of length d_{mics}, one side of length d and an angle θ between them. In order for θ to be calculated, the following equation is used:

$$\theta = \arctan(\frac{d}{d_{mics}})$$

θ equals the angle between the device and the sound origin. θ ranges from $0°$ to $90°$ to $180°$ as the source is moved from the right, to the middle, towards the left of the array.

Given the delay t_{delay} in time between the moment the sound hits microphone 1 (t_1) and the moment it hits microphone 2 (t_2), with $t_{delay} = t_1 - t_2$ d can be derived as follows:

$$d = t_{delay} \cdot V_{sound}$$

where V_{sound} is the speed of sound, 343 m/s

The challenge for this project is to find t_{delay} given just the raw microphone samples from ADC of the microcontroller. This process is called Time-Delay-angle-Of-Arrival (TDOA) analysis.

To estimate t_{delay}, we can use cross-correlation of two signals. Using cross-correlation, the presence of one signal can be detected within another signal. It is a measure of similarity between two signals, as one signal is delayed with respect to the other. For discrete, real-valued, signals, cross-correlation between signals x and y is defined as follows:

$$(x \star y)[n] \triangleq \sum_{m=-\infty}^{\infty} x[m] \cdot y[m+n]$$

This yields a new signal, of which the n corresponding the highest value is the delay between the signals for which they were most similar. Therefore, the delay between the signals x and y in terms of number of samples, n_{delay} can be found with

$$n_{delay} = \underset{n}{\operatorname{argmax}}\ (x \star y)[n]$$

Given a fixed sample period T_s used for both signals, t_{delay} can be calculated with

$$t_{delay} = n_{delay} \cdot T_s$$

t_{delay} has a maximum at $\theta = 180°$ and a (negative) minimum at $\theta = 0°$. Therefore, the calculations can be optimized by only calculating the cross-correlation for a limited range of n. The upper limit of n, n_{max}, can be found by

$$n_{max} = \frac{d_{mics}}{T_s \cdot V_{sound}}$$

The minimum is found by $n_{min} = -n_{max}$. Given these limits, the length of the cross-correlation signal can be reduced drastically:

$$(x \star y)[n] \triangleq \sum_{m=n_{min}}^{n_{max}} x[m] \cdot y[m+n]$$

If the wave length λ is no greater than d_{mics}, using the limited cross-correlation also makes the cross-correlation suitable for periodic sigals: the signals will only optimally lined up for one value of n. Therefore, a suitable value for d_{mics} needs to be found in order for the device to be able to work with typical mosquito buzzing sound, keeping in mind its periodicity. This is done by analyzing the Fourier transform of a mosquito buzzing sound fragment[10], as from it can be obtained the sound's fundamental frequency f_0. With f_0 determined, λ, and therefore d_{mics}, can be found with

$$\lambda = \frac{V_{sound}}{f_0}$$

This completes the set of calculations needed to calculate θ given two signals. In this project, the before calculations were implemented in order to find a suitable value for d_{mics}, and for the TDOA analysis to be done efficiently enough for them to be done on-device in real time.

3 Evaluation criteria

In order to indicate whether the device can locate a mosquito successfully, the following goal is defined. Firmware should be able to calculate the azimuth angle of a sound origin within a 10 degree error margin 80% of the time, when presented with a predefined sine wave as sound signal, as well as a real mosquito sound fragment.

4 Setup

To find a suitable value for d_{mics}, a Fourier transform of a mosquito sound fragment [10] was done. In figure 2, a plot of this transform was done. From the figure, it can be concluded that the sound signal in the analyzed fragment has a base frequency of about 367 Hz. Also, some strong harmonics can be found at $f_1 = 730\ Hz$ and $f_2 = 1102\ Hz$. The length of a full period is therefore determined as $\lambda = \frac{V_{sound}}{f_0} = \frac{343}{367} = 0.9\ m$. For our device to be able to work with this periodic mosquito sound, a value of $d_{mics} = 125\ mm$, which is significantly smaller than λ, is sufficient.

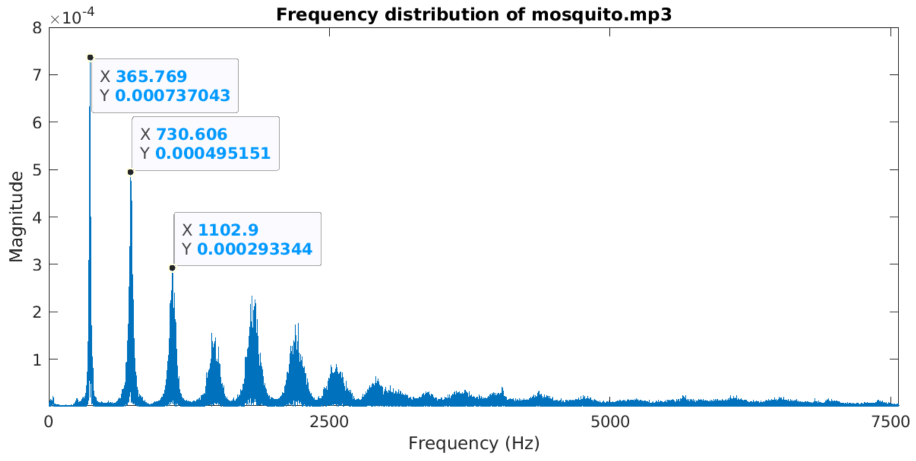


Figure 2: Fourier transform of mosquito.mp3

The device is controlled by a Nordic Semiconductor nRF52840 microcontroller [5], mounted on a nRF52840DK board, for which firmware is customly written in Rust [9]. The nRF52840 microcontroller is able to sample up to 8 analog inputs. Having four analog input channels enabled with an aquisition time of 5 μs, it is able to sample with a minimal sample period of $T_s = 22\ \mu s$. As the on-chip ADCs can be controlled by timer peripherals directly, the sample period can be configured with great flexibility. The nRF52840dk board can also relatively easily be set up for serial communication over USB [6], enabling running real-time analysis and graph plotting on a host computer for developmental ease. What's more, using the defmt [7] toolbox, data can be easily logged to the host device when attached to a debugger. This proves useful when obtaining raw samples or TDOA analysis results from the device.

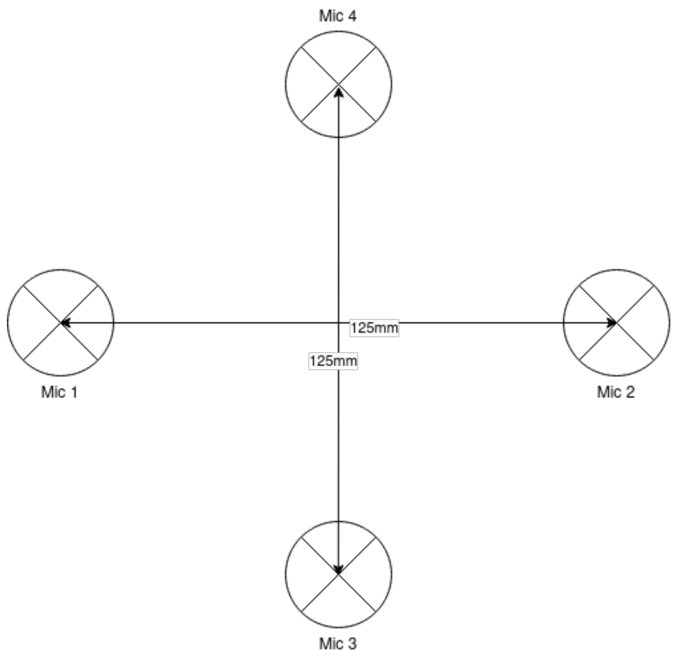


Figure 3: Microphone array dimensions

The device uses an array of four electret microphones, mounted in a '+'-shape. As can be seen in figure 3, every two opposite microphones are 125 mm apart. Using this setup, it becomes possible to estimate the angle from which the buzzing sound comes in both the horizontal and vertical axis. In this project, however, only the analysis of a single axis is benchmarked, as the accuracy of the second axis is expected to be very similar because of d_{mics} being equal for both axes.

The actual signals the device has to deal with are the analog microphone output voltages, being amplified by an op-amp integrated in each microphone module. As the device uses the ADC integrated in the nRF52840, the software handles raw measurements obtained from the ADC peripheral. The ADC peripheral can be assigned a block of memory with a customizable size. It can be configured to trigger an interrupt whenever the block is full, in order for the samples to be processed further. In this project, it is configured to produce 12-bit 2's complement integer values, with a sample period of $T_s = 37\mu s$. A suitable sample period was found by trial and error.

To manage the tasks the microcontroller has to fulfill in order to configure its peripherals, fetch and analyze samples and communicate, the device firmware is written with the RTIC framework [1]. This framework uses dynamic interrupt prioritization to schedule and pre-empt tasks. This framework is based om the Rust programming language, in which all of the firmware is written. Among many others, a trait of Rust is that it is very easy to write and compile platform-independent code. This enables us to test and validate firmware modules on a PC. The device firmware is available in the `./prototype/firmware` directory.

Another part of the project is the implementation of a command-line interface (CLI) application that can communicate with the device firmware over UART, the source of which is present in the `./prototype/cli` directory. Code in modules and libraries written in Rust can also be shared between multiple applications, enabling for easily defining a shared communication format, which has been done in this project. Its source code can be found in the `./prototype/format` directory. Using this communication format, the CLI is able to send commands to the device as well as receive raw samples from it. These raw samples can be logged to a file which Matlab is able to read using its 'readmatrix' command. Using the logged raw samples, Matlab scripts have been developed with which all calculations described in the problem specification are automated and can be found in the `./matlab` folder. These scripts are able to do the full TDOA analysis, given a file of raw measurements.

The Matlab script output is used as a basis for re-implementing the calculations in Rust, for them to be run on-device and in the CLI. These calculations are implemented in this `folley-calc` package, which is present in the `./prototype/calc` folder. This package is used by both the firmware and the CLI application to do analysis of raw samples, and contains unit tests that verify the calculations using output of the Matlab scripts for a recorded set of samples. As the `folley-calc` package is at the heart of the firmware application, some parts of this package is discussed in more detail in the following section.

5 Approach

The `folley-calc` package contains multiple functions that together implement all calculations needed to estimate the location of a sound-producing object given a set of samples. In listing 1, a snippet of the implementation of the cross-correlation is given. In this snippet, `SIGNAL_LEN` is the length of the signals `x` and `y`. In the firmware application, the value of `XCORR_LEN` is calculated by

$$XCORR\_LEN = 2 * n_{max} + 1$$

and is thus dependent on the distance between the two microphones the signals were obtained from and the sample period T_s. The result of the cross-correlation is stored in the `out` array, and the index for which the output signal has the greatest value, `argmax` is returned at the bottom. As indexing arrays with negative indices is not suppported in Rust, the actual lag n_{delay} is obtained by `argmax - (XCORR_LEN as isize / 2);`. As the value of `y[y_index]` might not always be defined, it defaults to 0.

```
//snip ...
let mut argmax = 0;
let mut max = 0;
for n in 0..XCORR_LEN {
    for m in 0..SIGNAL_LEN {
        let x_val = x[m] as i64;
        let y_index = (n + m) as isize - (XCORR_LEN as isize) / 2;
        let y_val = if y_index >= 0 {
            *y.get(y_index as usize).unwrap_or(&0)
        } else {
            0
        } as i64;
        out[n] += x_val * y_val;
    }
    if out[n] > max {
        max = out[n];
        argmax = n;
    }
}
argmax
//snip ...
```

Listing 1: Implementation of the cross-correlation of signals x and y in `folley_calc::xcorr_real`

With n_{lag} having been obtained, the next part is calculating the angle, given d_{mics} and T_s. This calculation involves geometrical calulations, which are typically slow on embedded devices. To mitigate this, a lookup table can be generated using the `folley_calc::gen_lag_table`. The generated table contains tuples in which n_{delay} is mapped to an angle in degrees. In listing 2 a snippet from an example lag table is given. The idea is that the lag table is generated only once at application start, and is used to look up angles corresponding to values for n_{delay}.

```
//snip ...
(-4, 99),
(-3, 97),
(-2, 95),
(-1, 93),
(0, 90),
(1, 88),
(2, 86),
(3, 84),
(4, 82),
//snip ...
```

Listing 2: Snippet of a lag table as generated by `folley_calc::gen_lag_table` with $T_s = 14\ \mu s$ and $d_{mics} = 125\ mm$

Listing 3 is an excerpt of the `folley_firmware::on_samples` firmware task, which receives the samples from ADC, calls the TDOA analysis function `folley_calc::calc_angle`, to obtain angles for two sets of microphones. The values for the constants passed to the `calc_angle` function are given in listing 4. From these values can be derived that the amount of samples in each of the compared signals is 1024. In other words, every TDOA analysis cycle is done with two signals containing 1024 raw samples, sampled with a sample period of $T_s = 37\ \mu s$. The firmware task prints the output using defmt to the console of the debugging host. An excerpt of example output is given in listing 5. During the evaluation experiment, only the values for the x-axis is reported, as the values for the y-axis are obtained in exactly the same manner. To analyze the measurements, the defmt output is stripped of anything not being the x-axis output. This yields a list of just the calculated angles for the x-axis.

```rust
//snip ...
let mut buf = [0i64; MAX_LAG];
let x_angle = folley_calc::calc_angle::<
    T_S_US,
    D_MICS_MM,
    MAX_LAG,
    SAMPLE_BUF_SIZE
>(
    &channels.ch1,
    &channels.ch2,
    &mut buf,
    ctx.resources.lag_table,
) as i32;
let y_angle = folley_calc::calc_angle::<
    T_S_US,
    D_MICS_MM,
    MAX_LAG,
    SAMPLE_BUF_SIZE
>(
    &channels.ch3,
    &channels.ch4,
    &mut buf,
    ctx.resources.lag_table,
) as i32;
defmt::info!("x: {}\t\ty: {}", x_angle, y_angle);
//snip ...
```

Listing 3: Snippet of `folley_firmware::on_samples`

```rust
pub mod consts {
    use folley_calc::max_lags_size;

    /// Sample period in microseconds
    pub const T_S_US: u32 = 37;
    /// Distance between two mics in millimeters
    pub const D_MICS_MM: u32 = 125;

    /// Size of a set of samples
    pub const SAMPLE_BUF_SIZE: usize = 1024;
    /// Amount of lags evaluated in the cross-correlation
    pub const XCORR_LEN: usize = max_lags_size(T_S_US, D_MICS_MM);
}
```

Listing 4: Snippet of `folley_firmware::consts`

```
25 INFO  x: 102    y: 90
26 INFO  x: 90     y: 90
27 INFO  x: 90     y: 90
28 INFO  x: 90     y: 90
29 INFO  x: 90     y: 90
30 INFO  x: 114    y: 90
31 INFO  x: 90     y: 90
32 INFO  x: 90     y: 90
33 INFO  x: 90     y: 90
34 INFO  x: 108    y: 90
35 INFO  x: 90     y: 90
36 INFO  x: 90     y: 90
37 INFO  x: 90     y: 90
38 INFO  x: 90     y: 90
39 INFO  x: 102    y: 85
```

Listing 5: Example defmt output of the `folley_firmware::on_samples` task

The actual evaluation test is done by placing the microphone array next to a speaker at a distance of about 50 cm, as can be seen in figure 4. The speaker is placed at angles of $\theta = 0°$, $\theta = 45°$, $\theta = 90°$, $\theta = 135°$, and $\theta = 180°$ from the horizontal microphone array. At each angle, the mosquito buzzing sound sample used in the Fourier analysis in section 4 is played from the speaker, as well as a sine wave noise of 1100Hz. The resulting log files are then analyzed in the Matlab function shown in listing 6. To it is passed the file name to analyze, as well as the angle θ at which the speaker was placed. The function calculates and prints the number of calculation outputs in the file, the average estimated angle, the standard deviation of the output, the amount and percentage of values that were within a 10° margin from the expected angle, and whether the test was passed for the current list of outputs. As a baseline, a set of ouputs is analyzed with the speaker turned off as well. The values obtained from this analysis are discussed in the next chapter.

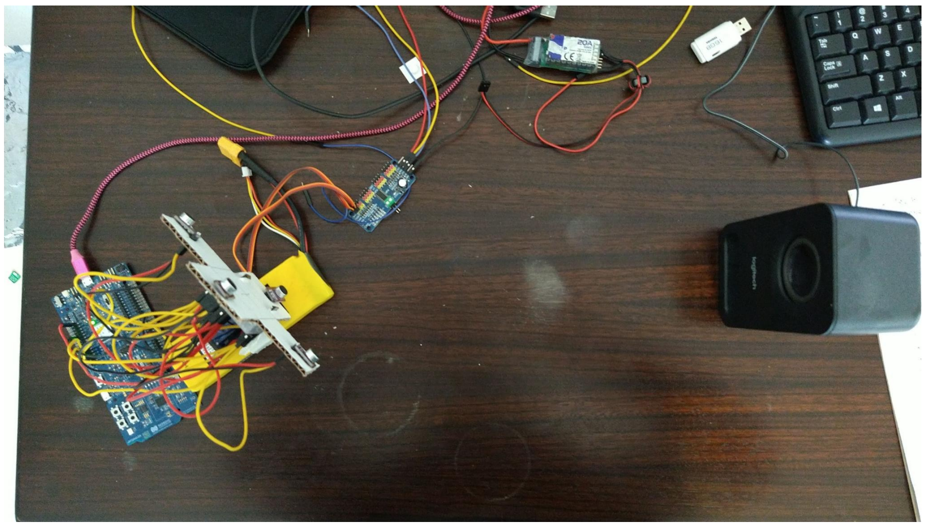


Figure 4: Setup for taking measurements

```
    function [ ...
    N, ...
    avg, ...
    passed_samples, ...
    pass_percentage, ...
    passed ...
] = analyze_measurements(file_path, expected_angle)
        [~, name, ext] = fileparts(file_path);
        file_basename = sprintf("%s%s", name, ext);
        samples = readmatrix(file_path);
        N = length(samples);
        avg = mean(samples);
        std_dev = std(samples);

        passed_samples = 0;
        for x = samples'
            if abs(x - expected_angle) < 10
                passed_samples = passed_samples + 1;
            end
        end

        pass_percentage = (passed_samples/N) * 100;
        passed = mat2str(pass_percentage > 80);

        fprintf("Number of samples:\t%d\n", N);
        fprintf("Average value:\t\t%g°\n", round(avg, 2))
        fprintf("Standard deviation:\t%g\n", round(std_dev, 2))
        fprintf("Percent within range:\t%g%%\n", round(pass_percentage, 2));
        fprintf("Pass:\t\t\t%s\n", passed);

        % snip
    end
```

Listing 6: `analyze_measurements.m` Matlab script

6 Results and Analysis

The results of the evaluation are shown in the tables below. In table 1, the results for the baseline measurement without any sound are shown. In the table, the value of N is the amount of calculation outputs that were analyzed, 101. In other words, the device took N times 1024 samples from the horizontal microphones, cross correlated the signals, and calculated the corrresponding angle from it. Without any sound playing, the device detects an angle of 90° with a standard deviation of 0. As there was no expected angle, data on passing the test was not recorded.

File name	N	Average (°)	Standard deviation
baseline.txt	101	90	0

Table 1: Results from testing with 1100Hz sine wave signal.

In table 2, the results of analyzing the 1100 Hz sine wave are shown. Only at angles $\theta = 45°$ and $\theta = 90°$, the test passes. For the other angles, the pass percentage is 0%, meaning none of the outputs were within 10° of the expected angle. The standard deviation is 0 or almost 0 for all angles, meaning that the calculations done by the device are consistent when presented a sine wave of 1100 Hz. Overall, however, the device does not meet the evaluation criteria.

File name	Expected angle (°)	N	Average (°)	Standard deviation	Passed samples	Pass percentage	Test passed
0_deg.txt	0	101	24	0	0	0 %	false
45_deg.txt	45	101	53	0	101	100 %	true
90_deg.txt	90	101	90	0	101	100 %	true
135_deg.txt	135	101	114.42	1.66	0	0 %	false
180_deg.txt	180	101	156	0	0	0 %	false

Table 2: Results from testing with 1100Hz sine wave signal.

Table 3 lists the results of analyzing the mosquito sound. Again, only two angles are estimated sufficiently, and the evaluation criteria are not met. We can see that the sandard deviation for the mosquito sounds are high at the extreme values for θ, but goes to zero as θ comes closer to 90°.

File name	Expected angle (°)	N	Average (°)	Standard deviation	Passed samples	Pass percentage	Test passed
0_deg.txt	0	101	32.07	20.21	0	0 %	false
45_deg.txt	45	101	56.29	8.24	75	74 %	false
90_deg.txt	90	101	90	0	101	100 %	true
135_deg.txt	135	101	128.72	11.03	91	90 %	true
180_deg.txt	180	101	146	22.81	0	0 %	false

Table 3: Results from testing with mosquito sound.

7 Conclusions

In project 'Folley', a device that analyzes mosquito buzzing noises and estimates the angle from which the sound comes was implemented. The core of the estimation procedure is cross-correlation of the signals obtained by the device's microphone array. Controlled by an nRF52840 microcontroller, the device uses a Rust-written implementation of the Time-Delay-angle-Of-Arrival analysis used to obtain signal delay, from which the sound's angle of arrival can be derived. In order for the analysis algorithm to be implemented and verified, a command line application that can receive and store raw measurements from the device, as well as a Matlab implementation of the analysis has been implemented. After implementing the device firmware as well as the analysis algorithm, the device analysis was tested by playing sounds from different angles and comparing the analysis output of the device with the expected output. Unfortunately, the device output does not meet the evaluation criterium. However, the output is remarkably consistent, especially for sounds coming from the front of the device.

References

[1] Real-time interrupt-driven concurrency. `https://rtic.rs/0.5/book/en/`. [Online; accessed 03-February-2022].

[2] Hongsen He, Lifu Wu, Jing Lu, Xiaojun Qiu, and Jingdong Chen. Time difference of arrival estimation exploiting multichannel spatio-temporal prediction. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(3):463–475, 2013.

[3] Henk Oordt. hdoordt/5liu0-2021. `https://github.com/hdoordt/5liu0-2021`.

[4] Nordic Semiconductor. nrf52840 dk. `https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk`, 2021. [Online; accessed 23-November-2021].

[5] Nordic Semiconductor. nrf52840 product specification v1.2. `https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.2.pdf`, 01 2021. [Online; accessed 03-February-2022].

[6] Martin D. Seyer Steven McDowell. *USB Explained.* Pearson, New York, 1998.

[7] Ferrous Systems. Knurling tools. `https://knurling.ferrous-systems.com/tools/`, 2020. [Online; accessed 02-February-2022].

[8] The MathWorks, Inc. Matlab, math. graphics. programming. `https://nl.mathworks.com/products/matlab.html`, 2021. [Online; accessed 23-November-2021].

[9] The Rust Team. Rust, A language empowering everyone to build reliable and efficient software. `https://www.rust-lang.org/`, 2021. [Online; accessed 23-November-2021].

[10] Zywx. Flying mosquito.wav. `https://freesound.org/people/Zywx/sounds/188708/`, 05 2013. [Online; accessed 02-February-2022].