Physics 361 - Electronics Lab

Doppler Effect - Final Lab

Jordan Ashley, Quintinne Madsen

May 2025

1 Introduction

The Doppler effect is the change in frequency observed when the distance between the observer and the source of the frequency-emitting device is changing. The goal of the project is to build a device that is:

- capable of displaying this effect
- portable
- simple enough to be used by middle-school aged children

The device must have a microphone to act as an observer of sound waves produced by an external source (speaker), as well as a way of displaying data.

2 Equations

- Speed of Sound in Air

$$v = 343 \, \frac{m}{s} \tag{1}$$

- Doppler Shift

$$f_o = f_s \left(\frac{v + v_o}{v + v_s} \right)$$

where f_o is the observed frequency, f_s is the source frequency, v_o is the velocity of the observer relative to the source and v_s is the velocity of the source relative to the observer (2)

- Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}, \quad \text{where } k \text{ is an integer ranging from 0 to } N-1 \tag{3}$$

- Radix-2 decimation-in-time Cooley-Tukey FFT

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k} \tag{4}$$

- Hamming window coefficients for a window of length $L = N + 1$

$$w(n) = 0.54 - 0.46 \cos\left(2\pi \frac{n}{N}\right), \quad 0 \leq n \leq N \tag{5}$$

- Variance

$$\sigma = \frac{\sum (x - \bar{x})^2}{n - 1} \tag{6}$$

3 Methods

3.1 Schematics

This project uses an Arduino UNO R3 to convert analog data taken from a MAX4466 microphone into digital data to be displayed on the liquid crystal display, LCD1602. The display's backlight is controlled by adjusting the dial of a 10 $k\Omega$ potentiometer. The device is powered by a 9 volt battery. All parts are assembled on a solderless breadboard.

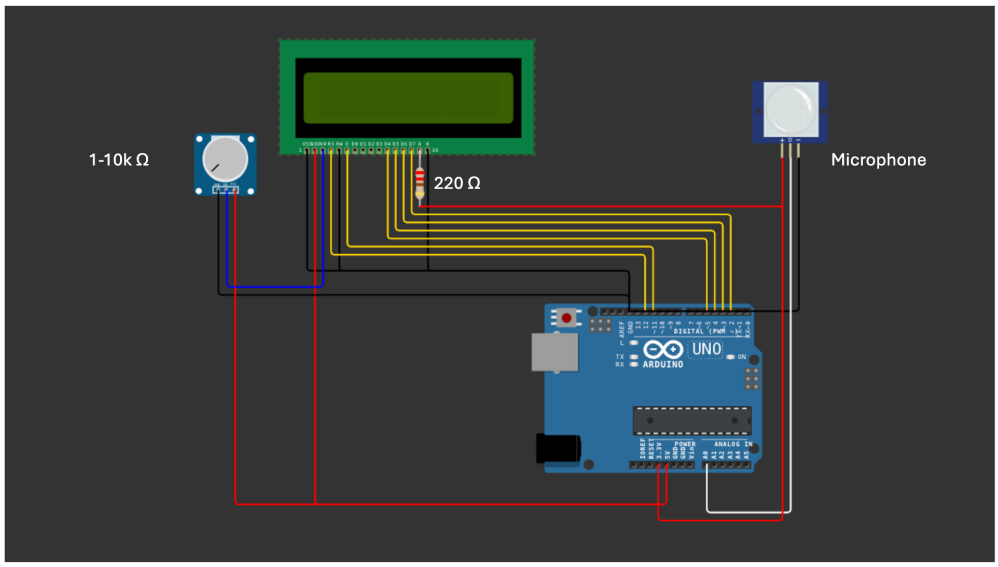

Figure 1: Wiring schematic of the device

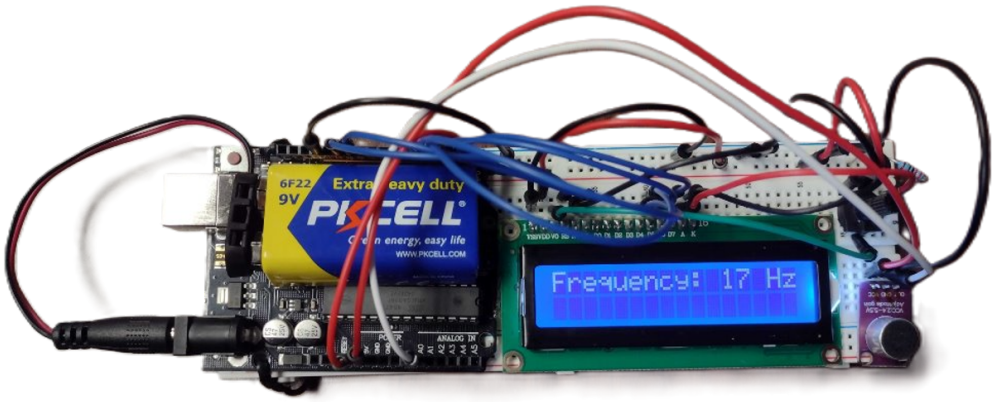

Figure 2: Photo of device

3.2 Commentary on Software Components

GitHub repository:

`https://github.com/j-s-ashley/phys361/tree/main/doppler`

A significant portion of the challenge of this experiment came from converting the analog data recorded by the microphone into interpretable frequency data. This was accomplished by using a fast Fourier-transform (FFT) with Hamming windowing to convert our time-dependent analog data to frequency in hertz.

3.2.1 Libraries

The `LiquidCrystal` library enabled the use of the LCD1602 module. A LiquidCrystal class object called `lcd` in line 40, which takes arguments specifying what Arduino pins are connected to the display.

The `arduinoFFT`[1] library enables the use of a pre-built fast Fourier transform (FFT), a class object of which is constructed in line 46 and built in the `runFFT()` function in lines 71-116.

The library itself seems to use the radix-2 decimation-in-time Cooley-Tukey FFT algorithm, which "rearranges the DFT of the function x_n into two parts: a sum over the even-numbered indices $n = 2m$ and a sum over the odd-numbered indices $n = 2m + 1$"[2], as shown in Equation (4). This inference was based on the bitwise swaps performed in lines 80-90 of the source code, arduinoFFT.cpp, as well as the iterative breakdown of discrete Fourier transforms (DFTs) (see Equation (3)) of size N into smaller ones of size $\frac{N}{2}$ in lines 100-122. The FFT algorithm was applied using the Hamming windowing method, specified in line 36 of our Arduino sketch, which creates a bell-curve using coefficients calculated via Equation (5).

3.2.2 Code

```
1 /*
2 --- start of header ---
3 doppler.ino
4 Takes analog input from microphone
5 Displays via LCD on Arduino
6
7 Last edit:
8 2025/05/08
```

```
 9 by
10 Jordan Ashley @j-s-ashley
11
12   The circuit:
13  * LCD RS pin to digital pin 12
14  * LCD Enable pin to digital pin 11
15  * LCD D4 pin to digital pin 5
16  * LCD D5 pin to digital pin 4
17  * LCD D6 pin to digital pin 3
18  * LCD D7 pin to digital pin 2
19  * LCD R/W pin to ground
20  * LCD VSS pin to ground
21  * LCD VCC pin to 5V
22  * 10K potentiometer
23    * ends to +5V and ground
24    * wiper to LCD VO pin (pin 3)
25  * MAX4466 VCC pin to 3.3V
26  * MAX4466 OUT pin to A0
27  * MAX4466 GND pin to ground
28
29 --- end of header ---
30 */
31
32
33 #define FFT_SPEED_OVER_PRECISION // use reciprocal multiplication
       for division + bonus speedups
34 #include <arduinoFFT.h>
35 #include <LiquidCrystal.h>
36
37 // Initialize the library by associating LCD interface pins
38 // with the arduino pin number it is connected to
39 const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
40 LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
41
42 // FFT setup
43 uint16_t FFTrefresh = 50;
44
45 #define MIC_PIN A0           // set arduino pin A0 as mic input
46 ArduinoFFT<double> FFT;      // initialize FFT class object
47 const uint16_t samples = 128; // set max samples held
```

```
48 double vReal[samples];
49 double vImag[samples];
50 const double samplingFrequency = 4000.0; // Hz
51
52 double lastPeak = 0.0;
53 const double alpha = 0.8;    // smoothing factor
54 double peak = 0.0;
55
56 // Timing
57 unsigned long lastFFTTime = 0;
58
59 void setup() {
60   lcd.begin(16, 2);
61   lcd.print("Starting up...");
62 }
63
64 void loop() {                                                    //
       hold FFT to refresh rate
65   if (millis() - lastFFTTime > FFTrefresh) {
66     runFFT();
67     lastFFTTime = millis();
68   }
69 }
70
71 void runFFT() {
72   // Sample the microphone
73   unsigned long startMicros = micros();                         //
       timestamp in microseconds
74   for (uint16_t i = 0; i < samples; i++) {                      //
       limit samples taken
75     // Sample every 1000000 / samplingFrequency seconds
76     while (micros() - startMicros < (1000000.0 / samplingFrequency)
         * i);
77     vReal[i] = analogRead(MIC_PIN);
78     vImag[i] = 0.0;
79   }
80
81   // Remove DC offset
82   double mean = 0;
83   for (uint16_t i = 0; i < samples; i++) mean += vReal[i];
```

```
84    mean /= samples;
85    for (uint16_t i = 0; i < samples; i++) vReal[i] -= mean;
86
87    // Perform FFT
88    FFT.windowing(vReal, samples, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
          // weigh samples
89    FFT.compute(vReal, vImag, samples, FFT_FORWARD);                //
          run calculations
90    FFT.complexToMagnitude(vReal, vImag, samples);                  //
          compute magnitude
91
92    // Check signal strength
93    double maxAmplitude = 0;
94    for (uint16_t i = 1; i < samples / 2; i++) {
95      if (vReal[i] > maxAmplitude) maxAmplitude = vReal[i];
96    }
97
98    if (maxAmplitude < 50) {
99      displayFrequency(0);                                          //
            Too quiet
100     return;
101   }
102
103   // Calculate dominant frequency
104   double rawPeak = FFT.majorPeak(vReal, samples, samplingFrequency);
105   if (rawPeak < 60 || rawPeak > 2000) rawPeak = 0;
106
107   // Smooth
108   if (abs(rawPeak - lastPeak) < 20) {
109     peak = lastPeak;
110   } else {
111     peak = alpha * lastPeak + (1.0 - alpha) * rawPeak;
112   }
113   lastPeak = peak;
114
115   displayFrequency((int)peak);                                     //
          send frequency to LCD
116 }
117
118 void displayFrequency(int freq) {
```

```
119   lcd.clear();
120   lcd.setCursor(0, 0);
121
122   if (freq == 0) {
123     lcd.print("Frequency: ----");
124   } else {
125     lcd.print("Frequency: ");
126     lcd.print(freq);
127     lcd.print(" Hz");
128   }
129 }
```

4 Testing

4.1 Procedure

To test the device, we used a portable speaker to play a single frequency while moving the speaker relative to the microphone. Before testing the Doppler response directly, we first identified an appropriate test frequency.

Our first observation was that the frequency reported by the device was consistently lower than the frequency reported by the source, which we confirmed by using two separate speakers, each tested with two separate frequency generators. The results between these tests was very consistent, showing no noticeable change between speakers or generators. However we did note that higher frequencies showed a consistently lower difference between the source-reported frequency and the device-reported frequency (Figure 3). Further exploration of the data showed that the variance of the frequency difference, as calculated with Equation (6), was minimized around 500 Hz, as shown in Figure 4.

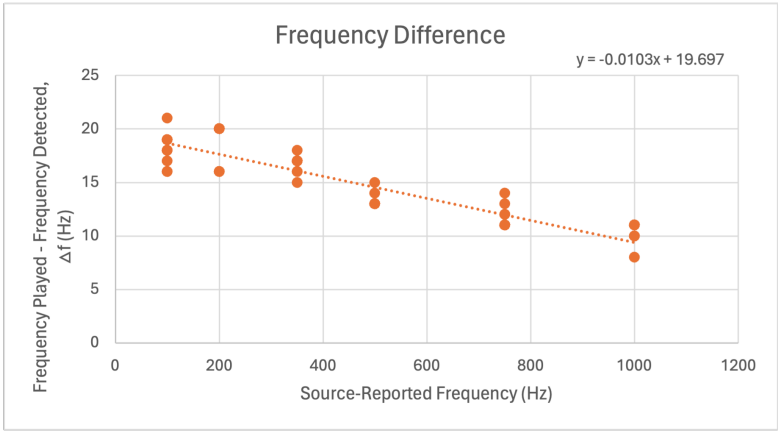

Figure 3: Difference between the source-reported frequency and the device-reported frequency

To directly test the device's sensitivity to the Doppler effect, we moved a speaker playing a constant 500 Hz tone relative to the microphone in three ways:

- source moving toward the device
- source moving away from the device
- swinging the source in circles in a plane level with the device

During each experiment, we first measured a stationary frequency before beginning any movement. Since our FFT calculations are iterative, this prevented an unrelated "ramp up" of the device's reported frequency being counted as part of our data and established a baseline to minimize the error introduced by the frequency reporting differences noted above.

We chose to keep the device stationary for all testing to minimize the potential for the microphone to pick up movement/jostling noise, as well as to avoid damaging the device.

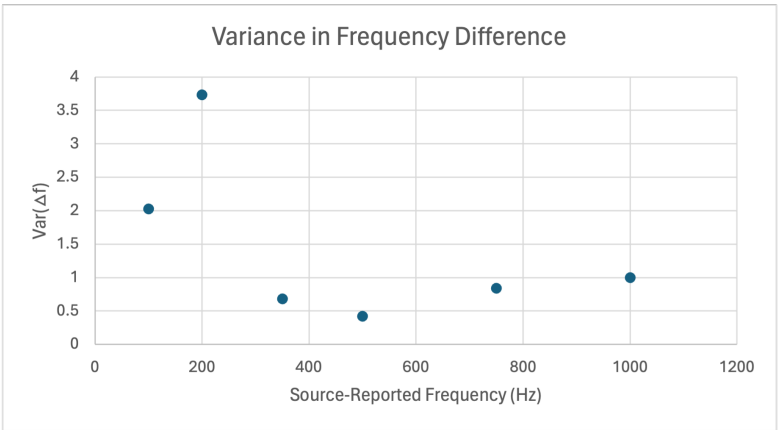

Figure 4: Variance in the difference between the source-reported frequency and the device-reported frequency

4.2 Data

The experiments showed data that matches the expected results, within a reasonable margin of error.

Negative v_s		
Stationary f	Highest f	Δf
487	491	4
487	495	8
487	492	5
487	492	5
487	489	2
485	490	5
485	486	1
487	491	4
487	488	1
487	495	8
487	491	4
487	491	4
487	488	1
487	494	7
486	491	5

Positive v_s		
Stationary f	Lowest f	Δf
487	484	-3
487	479	-8
487	479	-8
487	486	-1
487	485	-2
487	479	-8
487	486	-1
487	486	-1
487	485	-2
487	479	-8
489	488	-1
489	481	-8
489	485	-4
490	483	-7
495	487	-8

Each line in the tables above is a separate trial in which the speaker was moved either toward (negative v_s) or away from (positive v_s) the device by hand. For negative v_s, the speaker was thrown from near the device to approximately 3.43 meters away at speeds averaging 6 meters per second. For positive v_s, the speaker was held for the duration of the movement, restricting the velocity of the source (and, therefore, the change in frequency) to a slightly smaller number, but minimizing the potential for damage.

Plugging $v_o = 0,\ v_s \approx \pm 6\ \frac{m}{s}$, and $f_s = 500$ Hz into Equation (2) yields

$$f_o\left(-6\ \frac{m}{s}\right) = 508 \text{ Hz} \quad \Rightarrow \quad \Delta f = +8 \text{ Hz} \tag{7}$$

$$f_o\left(6\ \frac{m}{s}\right) = 491 \text{ Hz} \quad \Rightarrow \quad \Delta f = -9 \text{ Hz} \tag{8}$$

The frequency change recorded from the third trial, in which the speaker was attached to a cable and swung around in a plane level with the device, can be found in Figure 5.

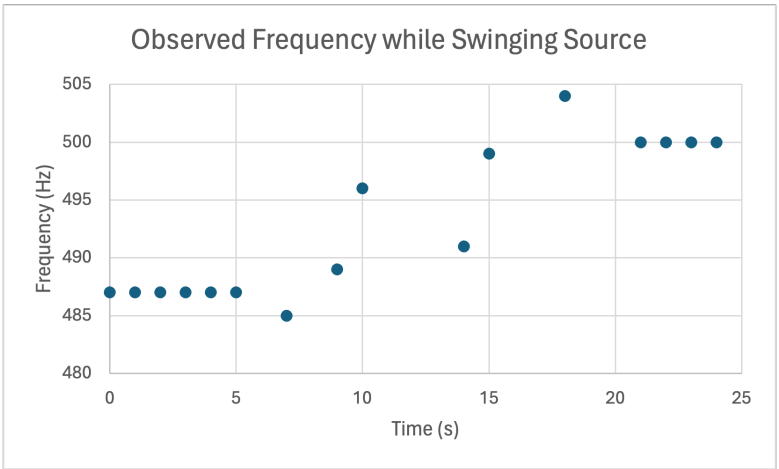

Figure 5: Frequency as a function of time for a source in circular motion

5 Conclusions

5.1 Results and Conclusions

The values are within a reasonable interval from our average recorded changes in frequencies, which were $\Delta f_{+v_s} = 4.27$ Hz and $\Delta f_{-v_s} = -4.67$ Hz. The

lower magnitude of the frequency changes compared to the theoretical results calculated in Equations (7) and (8) are very likely due to the iterative calculation used in the FFT, which is designed to minimize fluctuations in reported frequencies. Therefore, it makes sense that these changes would experience a "flattening." The bias toward more accuracy at higher frequencies may account for the overall increasing trend seen in the varied-motion trial plotted in Figure 5, which serves as a perfect visual for both the reliability and limitations of the device.

The Doppler effect is clearly seen in the consistent oscillations between higher and lower frequencies, but the high-frequency bias of the observed frequency makes nuanced calculations overly complex. However, these limitations do not nullify the overall success in meeting our project goals.

6

References

[1] Condes Brena, E., Overbohm, B., Lukken, C., FintasticMan, & MarScaper. (2024). ArduinoFFT. Fast Fourier Transform for embedded devices (2.0.4). Zenodo. https://doi.org/10.5281/zenodo.14195818

[2] Cooley–Tukey FFT algorithm. The radix-2 DIT case (accessed May 8, 2025). Wikipedia. `https://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm`