Workshop Exercise 2 Guide!

Welcome to the music note analyzer workshop activity! Here, we build aim to build a device that provides the information necessary to tune an instrument.

Goals

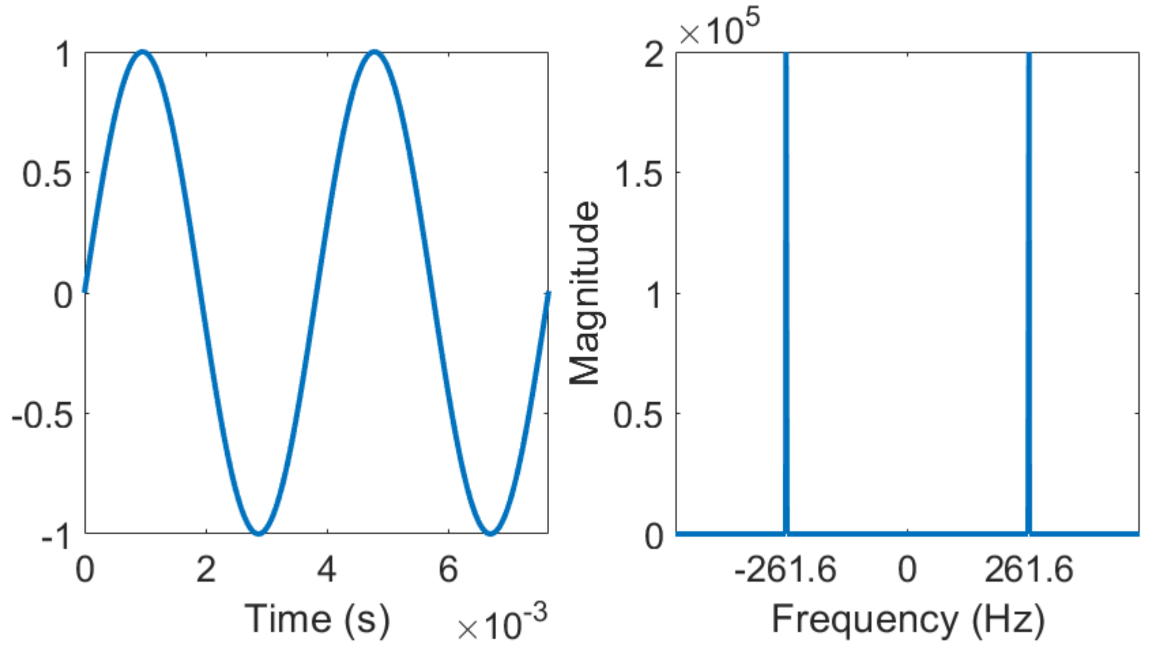
In this workshop, we hope to develop a conceptual understanding of how the frequencies of sound waves map to musical notes. Each instrument obtains its unique sound by producing sound waves at many frequencies. We will apply our knowledge of the Fourier Transform to uncover the dominant frequency in a sound wave, which ends up being the musical tone we perceive the sound to be at.

From this, we will compare this frequency to a known mathematical model, to determine how this frequency aligns with the primary music notes (i.e. A, A#, B, etc). Primarily, we are looking to convince ourselves that it is possible to take in any incoming sound signal, and break it down into its frequency components.

Theory

Frequency and Music Notes

We will use the "C" note as an example. If we create a sinusodal wave that oscillates as a specific frequency, we obtain two spikes from the Fourier transform (FT).



In this case, we created a 261.6 Hz wave, and get two peaks following the FT, at -261.6 Hz and +261.6 Hz. The negative values are a consequence of the mathematics, and aren't interpretable for sound. For this exercise, we will ignore them and turn our focus to values above 0.

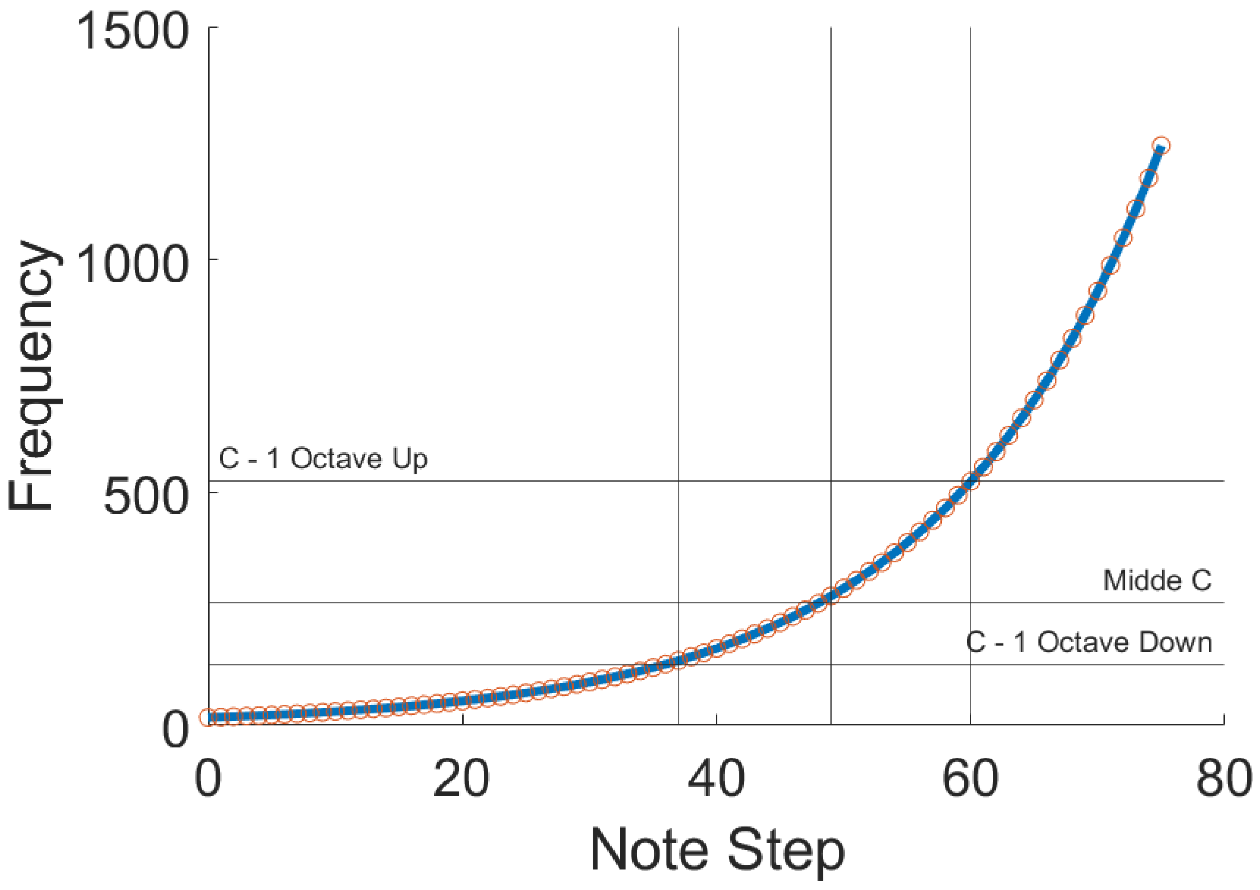
This is great! We can determine the frequency content of an incoming sound wave, but how does this map to musical notes?

We know that the frequency of musical notes changes exponentially, with base 2. We also know that the notes repeat every octave, with 12 different notes within an octave. If we start at some musical note f_0, we can determine the frequency of a musical note 'n' notes away using the following formula:

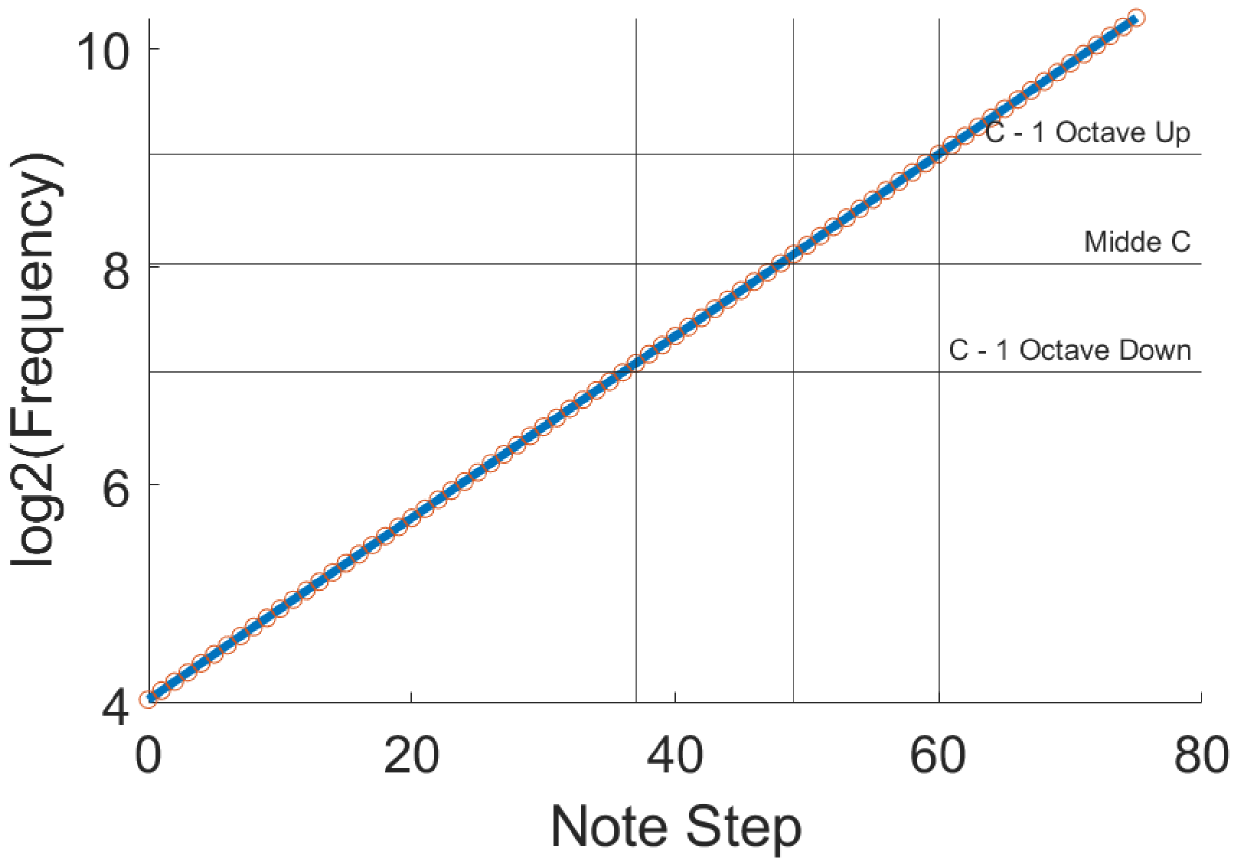
$$f_n = f_0 * 2^{(n/12)}$$

The means that the frequency of each note varies by a factor of $2^{(1/12)}$ or ~1.05946. If we know that note 'C' in Octave 0 has frequency 16.35 Hz, we can use this as f_0 to map to any other music note. For example, we calculate that the plotted 'Middle C' note is 49 musical note steps away from this base note.

If we want to see what this looks like as a plot:

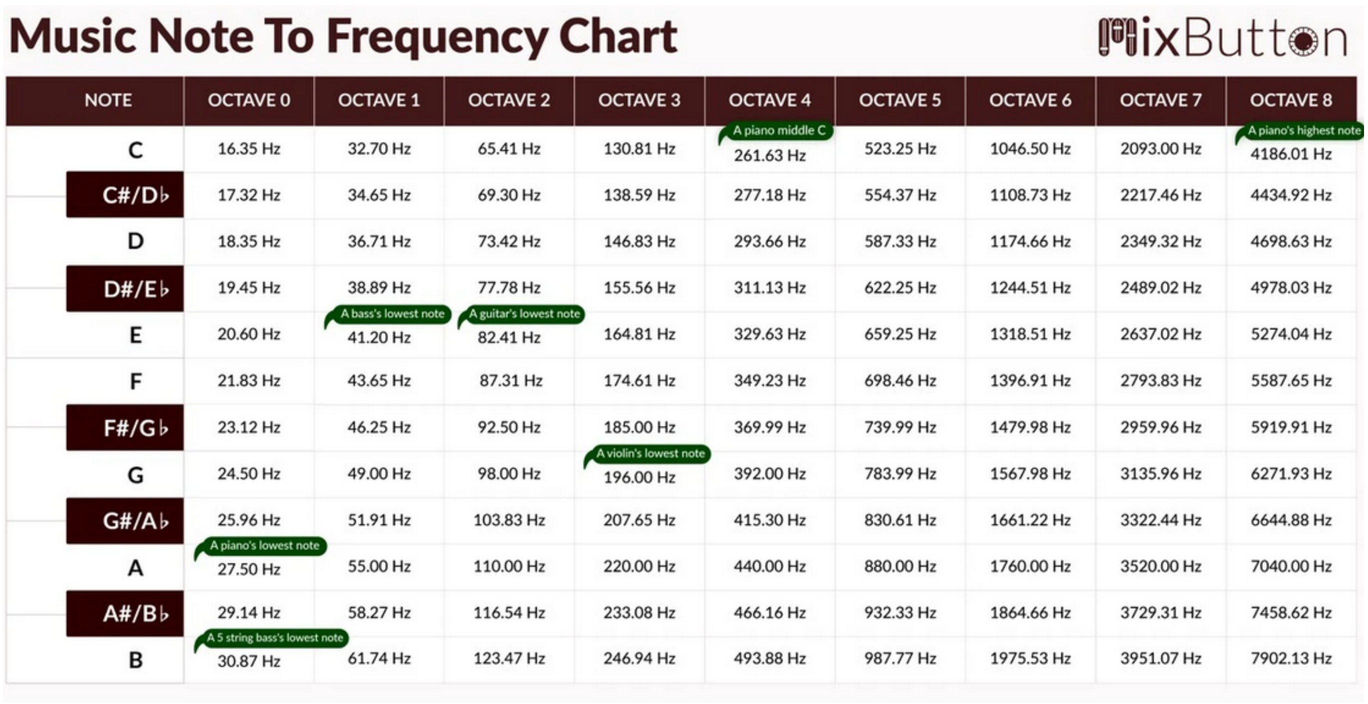


From this, we can clearly see the exponential change of steps in frequency. If we take log2 of both sides of the equation, then we should expect a linear result. If we plot that, we see:



And this is more so how we percieve a change in musical tones, which means our ears are senstive to sound waves in a non-linear manner, even if we percieve them linearly.

In chart form, we can see all the frequencies listed:



Music Note To Frequency Chart

NOTE	OCTAVE 0	OCTAVE 1	OCTAVE 2	OCTAVE 3	OCTAVE 4	OCTAVE 5	OCTAVE 6	OCTAVE 7	OCTAVE 8
C	16.35 Hz	32.70 Hz	65.41 Hz	130.81 Hz	A piano middle C 261.63 Hz	523.25 Hz	1046.50 Hz	2093.00 Hz	A piano's highest note 4186.01 Hz
C#/D♭	17.32 Hz	34.65 Hz	69.30 Hz	138.59 Hz	277.18 Hz	554.37 Hz	1108.73 Hz	2217.46 Hz	4434.92 Hz
D	18.35 Hz	36.71 Hz	73.42 Hz	146.83 Hz	293.66 Hz	587.33 Hz	1174.66 Hz	2349.32 Hz	4698.63 Hz
D#/E♭	19.45 Hz	38.89 Hz	77.78 Hz	155.56 Hz	311.13 Hz	622.25 Hz	1244.51 Hz	2489.02 Hz	4978.03 Hz
E	20.60 Hz	A bass's lowest note 41.20 Hz	A guitar's lowest note 82.41 Hz	164.81 Hz	329.63 Hz	659.25 Hz	1318.51 Hz	2637.02 Hz	5274.04 Hz
F	21.83 Hz	43.65 Hz	87.31 Hz	174.61 Hz	349.23 Hz	698.46 Hz	1396.91 Hz	2793.83 Hz	5587.65 Hz
F#/G♭	23.12 Hz	46.25 Hz	92.50 Hz	185.00 Hz	369.99 Hz	739.99 Hz	1479.98 Hz	2959.96 Hz	5919.91 Hz
G	24.50 Hz	49.00 Hz	98.00 Hz	A violin's lowest note 196.00 Hz	392.00 Hz	783.99 Hz	1567.98 Hz	3135.96 Hz	6271.93 Hz
G#/A♭	25.96 Hz	51.91 Hz	103.83 Hz	207.65 Hz	415.30 Hz	830.61 Hz	1661.22 Hz	3322.44 Hz	6644.88 Hz
A	A piano's lowest note 27.50 Hz	55.00 Hz	110.00 Hz	220.00 Hz	440.00 Hz	880.00 Hz	1760.00 Hz	3520.00 Hz	7040.00 Hz
A#/B♭	29.14 Hz	58.27 Hz	116.54 Hz	233.08 Hz	466.16 Hz	932.33 Hz	1864.66 Hz	3729.31 Hz	7458.62 Hz
B	A 5 string bass's lowest note 30.87 Hz	61.74 Hz	123.47 Hz	246.94 Hz	493.88 Hz	987.77 Hz	1975.53 Hz	3951.07 Hz	7902.13 Hz

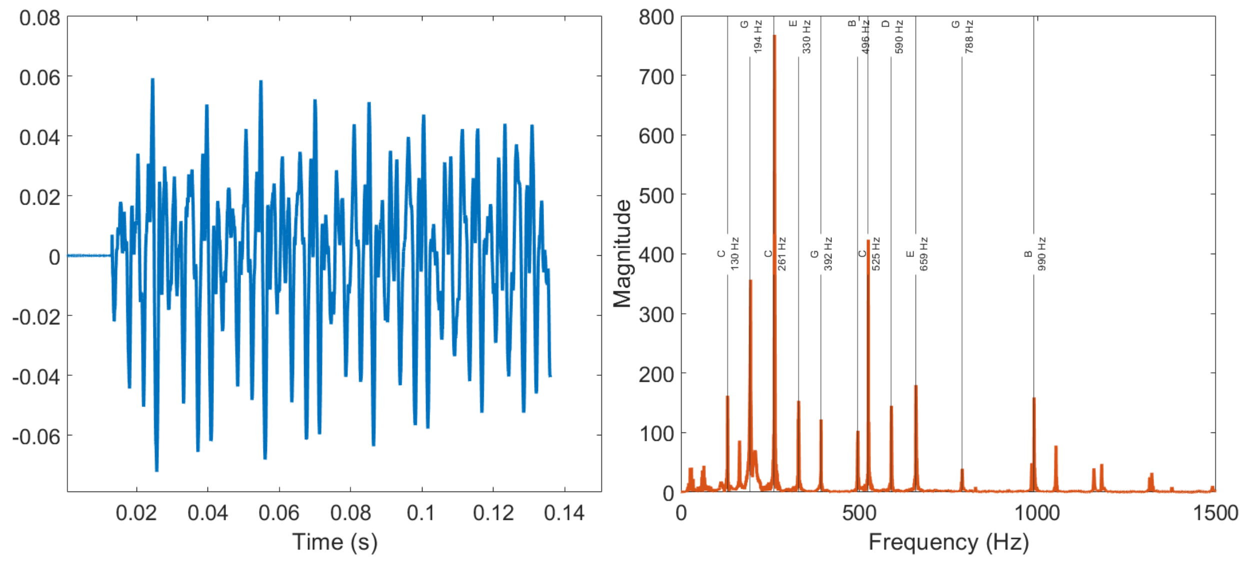
The chart points out a few reference values:

- Middle C for the piano is 261.63 Hz
- The A in octave 0 has a frequency of 27.5 Hz, and is the lowest note on a piano

- The piano's highest note is a C in the 8th octave, at 4186.01 Hz
- A guitar's lowest note is the 2nd octave E at 82.41 Hz

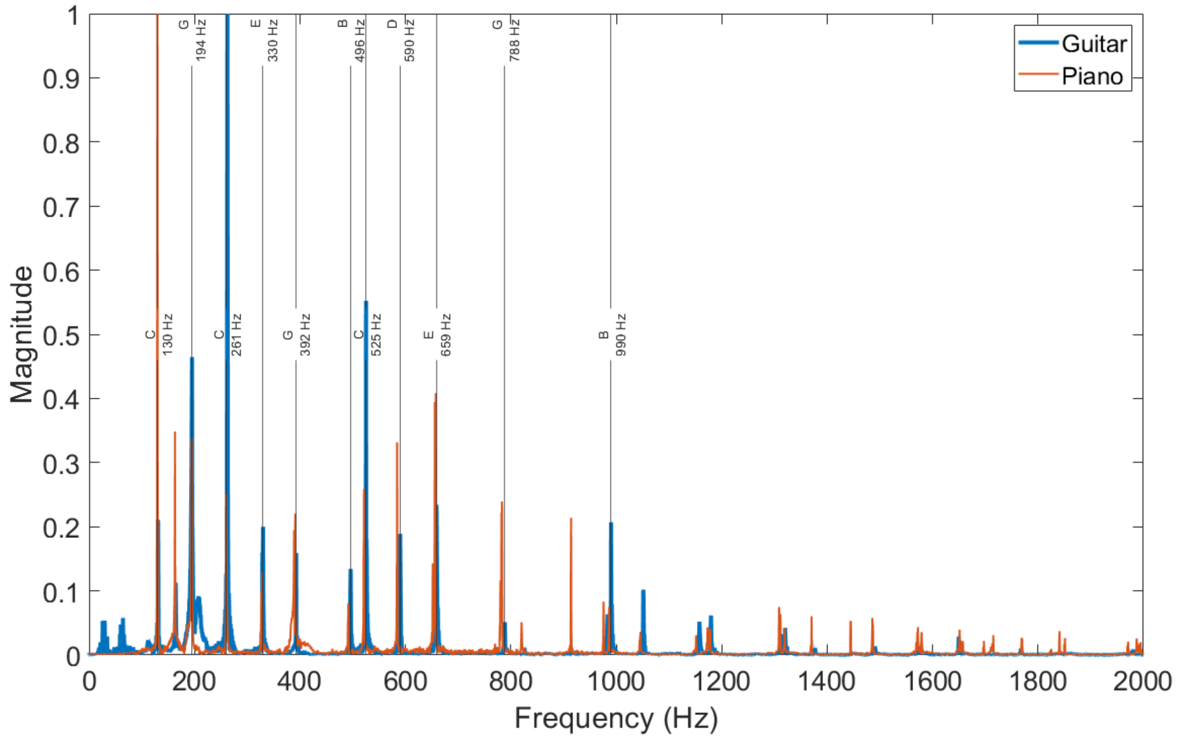
Musical Instruments

To obtain a better appreciation of what separates a 'C' chord in piano and guitar, we will look at the Fourier transform of each. The C Major chord contains the following notes: C, D, E, G, A and B. The soundwave and Fourier transform of a guitar C major chord looks like:



The input audio looks quite random, but from the FT of that signal, we can see has discrete peaks. If we look into the values of these peaks we see that they align with different musical notes! We perceive this to be a 'C' note, because it is the largest peak in the FT. We won't go into the music theory of what other notes best pair to make a nice sounding 'chord'.

Let us compare the frequency spectrum from a guitar to one from a piano. The piano note is one octave lower to help us visualize differences.



We have a primary difference of 1 octave in the C notes. But you can see that despite the piano being a lower note, it has more high frequency content that provides its characteristic sound.

Hopefully this helps illustrate the usefulness of the Fourier transform. Now we have a quantitative metric to be able to compare two signals, that provides meaningful interpretation.

Bringing It Together

Hopefully from the above, we can see:

- musical instruments differ in sound based on the contributions of additional frequencies
- the musical note we 'hear' is the dominant frequency in the wave
- we can use a mathematical formula to determine the frequency of any musical note

We can make our 'note detector' by pre-computing the frequency values of all the notes in a frequency range of interest (~10-5000 Hz), and then compare our detected frequency to find the closest musical notes.

Hardware setup

To begin, you should set your hardware up as illustrated in the README.md file.

Coding Steps:

1. Start by selecting the colour scheme you want by uncommenting only one section in Lines 55-85. This is done by toggling the comments (quick way is to highlight text and press CTRL+ '/'). The default is the rainbow colour scheme.

2. Next, upload W2_detectNote.ino to the ESP32 with all the default values. You may need to check that you have the right COM port selected. Hold down the boot button on the ESP32 while uploading.

3. If the upload was successful, and all the parts are properly connected, then you should see the lights working now if you are in a noisy environment. If not, make some noises to test it!

4. At the top of the arduino file, we have set up a user configuration zone. The main variable of interest is `NOISE_PEAK`. This will be a different noise value compared to W1. This noise variable is so that we do not have the LEDs light up when we do not purposely play a sound. This effectively blocks out background noise. If you are in a noisy location, increasing the `NOISE_PEAK` variable can help. You can guess and check... Or we can have the ESP32 print us the data values from the microphone to the Serial Monitor. (see next step!)

5. Go to line ~127 and look for:

```
// Serial.print(peakF, 2); // Frequency of Peak -> 2 is for 2 decimal points.
// Serial.print(", ");
// Serial.println(peakM, 2); // Magnitude of Peak
```

5. Continued... Remove the `//` to uncomment these 3 lines. Re-upload the sketch to the ESP32. Go to the serial monitor and watch the values popping up on the screen. If you are not making any noise to the microphone, what is an average low value for peakM? Take that value, add ~100, and set that as the `NOISE_PEAK` variable. The peakF variable tells you what the peak frequency it is detecting, we will look at this later! Upload that update to the ESP32 and see how it now responds to noise at the bottom (quiet) end. Repeat as necessary so that the lights mostly start to turn on when you deliberately make noise.

Testing Steps:

The hard part is done and now the fun begins! We have the hardware working, and the code configured for our setup, now we want to put it to the test. We have a few audio files to test:

1. We can start by playing the `middle C` file. A single note (created synthetically of course) will consist of a sine wave at a specific frequency. As we play this note, we should see a single bar light up. Test that you get different bars lighting up for `A_octave2.wav` and `G_octave5.wav`. Check the serial montior to see what frequency is displayed, it will be the first number on each line printed. How does it compare to the chart above?

2. In W1 the `frequencySweep.wav` file scanned its way across the LED matrix. How do we expect that behaviour to change under this configuration? Test it to see if the results match your expectation!

3. The files tested above only played a single note at each point in time. What if we played multiple notes at once, as a typical instrument would? Included are the `GuitarC.wav` and `PianoC.wav` files that the Fourier transforms were plotted above. If they are both 'C' notes, they should both light up the same bar. Is that what you observe?