Workshop Exercise 1 Guide!

Welcome to the spectrum analyzer workshop! In this workshop, we build a spectrum analyzer, where the LED vertical bars correspond to the sound within a particular frequency band (ie 400-1000 Hz). The amount of squares lit up with each bar will correspond to the relative amount of sound in that frequency band.

Goals

In this workshop, we hope to develop a conceptual understanding of how the Fourier Transform works. 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.

In order to achieve this, we will also have to become familiar with breadboards, jumper cables, the ESP32 and the LED Matrix.

Coding Steps:

1. Start by uploading W1_spectrumAnalyzer.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.
2. 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!
3. At the top of the arduino file, we have set up a user configuration zone. The two main variables of interest are the `AMPLITUDE` and `NOISE`. The amplitude sets the value for what signal from the microphone reaches the top of the LED. The noise variable defines the bottom. If you are in a noisy location, increasing the `NOISE` 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.
4. Go to line ~87 and look for `//Serial.println(vReal[5]);`. Remove the `//` to uncomment the line, and the re-upload the sketch to the ESP32. Go to the serial monitor and watch the values popping up on the screen. Is there an average value at the low end that seems consistent? Take that value, add 20-50, and set that as the `NOISE` variable. 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.
5. Now we want to set the peak. First, lets comment out the line from before, by adding `//` back in front of it. Now jump to line ~125 and look for: `// if (band == 3) Serial.println(barHeight);` Uncomment this line, and reupload to ESP32. Just like in the last step, we are watching the serial monitor, but now, looking for the maximum values we see. This time, we want to make some noise (but not too much!), so that we know what we want the value to be for full illumination. Once you find a value, set that as the `AMPLITUDE` variable. Upload that update to the ESP32 and see how it now responds to noise at the top (loud) end. Repeat as necessary so that the lights respond the way you would like them to. You may want to go back to step 4 if you feel like the `NOISE` could be further adjusted. It doesn't need to be perfect, as the level of noise in the room will be changing. You just want it set well enough that you can see how it responds to new incoming 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.
2. Ok, so a single bar could have been a fluke. To test this some more, lets play a `frequency sweep` file. This should have the sound going from a low frequency to a high frequency. If our theory works, then we should see a bar light up on one side, and move across the LED matrix as it moves to a higher pitch.
3. Enough of the synthetic waves! What about real audio? What gives a guitar the sound of a guitar, despite playing the same musical note as a piano? To better visualize this, we may want to adjust the frequencies that the bars of our LED correspond to. While we can hear from ~20- 20,000 Hz, most of the instrument and vocal frequencies are from the 20-10,000 Hz range. The first 5 octaves are all below 1,000 Hz. To better see how we can pick up small differences in notes, copy the following into your code from lines ~98 to ~114, replacing the `if` statements that are there:

```
if (i>3   && i<=4  ) bandValues[1]  += (int)vReal[i];
if (i>4   && i<=5  ) bandValues[2]  += (int)vReal[i];
if (i>5   && i<=6  ) bandValues[3]  += (int)vReal[i];
if (i>6   && i<=7  ) bandValues[4]  += (int)vReal[i];
if (i>7   && i<=9  ) bandValues[5]  += (int)vReal[i];
if (i>9   && i<=11  ) bandValues[6]  += (int)vReal[i];
if (i>11   && i<=14  ) bandValues[7]  += (int)vReal[i];
if (i>14   && i<=18  ) bandValues[8]  += (int)vReal[i];
if (i>18   && i<=22  ) bandValues[9]  += (int)vReal[i];
if (i>22   && i<=28  ) bandValues[10]  += (int)vReal[i];
if (i>28   && i<=35  ) bandValues[11]  += (int)vReal[i];
if (i>35   && i<=44  ) bandValues[12]  += (int)vReal[i];
if (i>44   && i<=55  ) bandValues[13]  += (int)vReal[i];
if (i>55   && i<=69  ) bandValues[14]  += (int)vReal[i];
if (i>69             ) bandValues[15]  += (int)vReal[i];
```

3. Continued... This changes the bins to be sensitive to frequencies between 100 and 3,000 Hz (2nd to into the 7th Octave). Before you upload this change, try talking into your microphone and see how the bars are clustered together. How does that change with the new frequency range of the bars?
4. Now we want to do what we set out to do in 3. Lets play and compare a C note from a guitar and a piano. How do the bars look?

In workshop activity 2, we will see that we can use the Fourier Tranform to detect music notes!