AUDIO-REACTIVE LED LIGHT SEQUENCER

Alin Kenworthy, Juan Estrella

Univeristy of Rochester, Department of Audio and Music Engineering

ABSTRACT

The visualization of sound is a useful and often overlooked tool for both the student attempting to understand abstract audio phenomena, and the consumer, pursuant of a more holistic musical experience. Sound visualization is also a very relevant field for the deaf and hearing impaired.

While there are many existing libraries and software applications that offer simple and intuitive musical/visual interface, and processing, (such as Pure Data, MaxMSP, and MATLAB's Audio System Toolbox), here we develop our own algorithm, from the ground up, for the musical control of lights, by utilizing and expanding upon methods learned in class, taught by Professor Zhiyao Duan.

This project is the first step along our path to create a standalone, real-time, audio reactive light strip, with downloadable applications for students and consumers.

By posting our code online we hope to contribute to the limited documentation for MATLAB to Arduino interfacing and LED control. All functions are fully commented, with many user adjustable parameters.

Index Terms— MATLAB, Arduino, LEDs, lights, sequencers, FFT, processing

1. INTRODUCTION

Here we present 3 separate MATLAB applications for LED control based upon musical frequency, amplitude and onset. The signal chain is as follows:
After a .wav file or mp3 is loaded into MATLAB and processed, controlling information is converted via the MATLAB To Arduino Add-On, and sent to an Arduino board. The Arduino then uses this information to control the intensity, color and state of 60 LEDs on a NeoPixel strip.
Though the conversion from MATLAB to Arduino introduces some latency and timing issues, the code is kept in MATLAB because results can be seen and analyzed immediately without compiling.

The amplitude application developed here creates a digital VU meter, centered in the middle of the LED strip, and the lights pulsate outward to the ends of the strip and back in, correlating to the amplitude of the left and right channels.

The frequency application assigns to each light a 333 hertz bin, progressing in a linear fashion from left to right, from 20Hz to 20,000Hz. As the power in a bin increases, the associated LED changes in color, from off, to cooler colors such as blue, to warmer colors, such as red.

Finally, our Onset Detection Program uses the same 'frequency bins' as the Frequency application, but instead of analyzing each bin separately, bins are compared, and if a sufficient rate of change is measured then a note is considered to be triggered, and the LED of the associated frequency bin is illuminated.

2. HARDWARE

2.1. Arduino and MATLAB

The Arduino and MATLAB are easily interfaced when set up properly. There are a few packages to install to MATLAB and a single USB connection required to connect the Arduino to the computer host. The processing is done in in MATLAB and the Arduino is responsible for carrying out the commands dictated by the user through the code. This is because MATLAB will upload a server to the board when the Arduino object is created by the user.

2.1. Arduino and NeoPixel LEDs

The NeoPixel Light Emitting Diode strips are a fully customizable, programmable, tool that can be controlled using an Arduino. In addition, the NeoPixel command library may be imported to MATLAB using an addon package. The setup for LEDs to be connected to the Arduino is very simple. Once the LEDs, Arduino, and MATLAB are all connected correctly, the process of programing the lights is seamless.

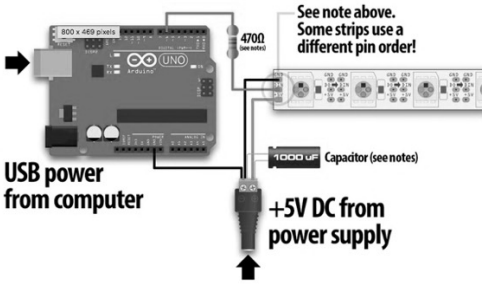

Figure 1: The connections to be made for proper interfacing between the Arduino and NeoPixels

3. AUDIO PROCESSING

3.1. Preparing Audio

Audio used in this program are imported as an audio file and stored in an array. The array is then separated into frames for processing. The user may select the size of the frame to be processed by editing the number of frames to be take ever one second of audio. To account for incomplete frames at the end of the audio file, zeros are padded on the end such that the total number of samples after padding is divisible by the sampling rate. This allows for easy processing in later sections of the program. The padding process is carried out before frames are created. Now that the audio file has been prepared; that is, it has been padded and framed; it is ready for one of three processing functions.

```
for y= x*binSize:((x+1)*binSize)-1
       frame(counter)= audio(y,chan);
       counter= counter+1;
end
```

Code Snippet 1: Creating a frame for processing

3.2. Amplitude Analysis

Amplitude analysis is used to extract the amplitude of the waveform for the duration of the file. The audio file contains integer ranging from -1 to +1. To prepare the audio's amplitude to be displayed on the light strip, the absolute value of the audio is first multiplied by the number of LEDs that will represent each channel. For example, if there is 30 LEDs that will represent each channel, a value of 1 in the audio file will translate to 30 and thus 30 LEDs will light up as the amplitude is a maximum. Likewise, if the amplitude is -.5, the number of LEDs that ill light is 15.

For each frame analyzed, the average amplitude is taken. This value is rounded up. This is because the likelihood of every sample in a frame being equal to 30 is highly unlikely. The result of the processing is an array of integers with values ranging from 0 to 30. The dimension of the array is 2 (channels) by the number of frames per second multiplied by the length of the song. The calculation is further simplified since we know that in the previous padding section the audio was padded to the next second. For example, we know a 3.5 second audio clip will have the dimension of 2 x (frames per second * 4).

```
bins(x,chan)= ceil(mean(frame));
```

Code Snippet 2: Calculating the average amplitude of a frame

3.2. Frequency Spectrum Processing

The frequency spectrum processing begins in a similar fashion as the amplitude spectrum processing, with the exception of the scalar multiplication. For each frame of audio, the Fast Fourier Transform is performed. Because this method results in a two-sided result; there are negative and positive values, the negative values are removed.

The FFT taken over a frame would normally return a matrix with the dimension equal to the length of the frame being taken. For this project, the dimension of the FFT being returned after the negative values are removed should be equal to the number of LEDs on the light strip. This is because we want to represent the frequency spectrum for each frame using the number of LEDs on the strip. For example, we used a strip with 60 LEDs. To return an FFT with 60 bins, a 120-point FFT is taken. Once half of the result is removed; the negative indices are removed a previously discussed; we are left with 60 bins containing the magnitude for (1/60) of the 20Hz-20kHz range of the audio signal. Each bin covers approximately 333Hz.

For each frame, the 60 bins are scaled arbitrarily to be sorted. The magnitudes in the 60 bins are replaced with an integer between 1 and 8. The integers being placed in each bin are a result of the magnitude satisfying a condition that considers the strength of each bin. For example, a bin where the magnitude is rather large will return a higher number, say 7 or 8, than a bin with a weaker magnitude, which might only return 2 or 3. The final product produced by the frequency analysis function is a Total number of frames x 2 x 60 array. Each frame has the 60 scaled magnitudes for each of the two channels of audio.

```
Y = fft(frame,(numLEDs*2));
TS = abs(Y/(numLEDs*2));
OS= TS(1:(numLEDs*2)/2+1);
OS(2:end-1) = 2*OS(2:end-1);
OS= OS(1:numLEDs);
```

Code Snippet 3: Finding the FFT of one frame

3.3. Onset Detection Processing

Onset detection processing is similar to the process of frequency analysis. The difference is that instead of scaling

frames for all sixty bins, the strongest difference between the bins of two audio frames is analyzed for its strength, rather than analyzing all sixty bins individually.

In this process the FFT of two conservative frames are taken individually. Two 2x60 matrices are created. Next, the strongest magnitude for each bin's left and right channels are found and put into a matrix. We now have frame A's 1x60 maximum strength matrix for each bin and frame B's counterpart. At this point we want to find the derivative of the amplitude, so the difference between the two maximum value matrices is taken. There is now one, 1x60 matrix with the difference between the two frames' magnitudes for each of the 60 bins. The largest value in this matrix is assumed to be the onset of a new note. For example, in frame one 440Hz bin may have magnitude of 0. In the next frame the note may be sounded and now have a magnitude of 60. This could be true about several bins if a chord is being sounded. To ensure the note is actually beginning in the frame being analyzed, a threshold is set to which the magnitude change must be greater than for the LED to be illuminated. The result of this function is a number of frames x 60 matrix which holds markers for if/when a note is sounded in each frame.

```
% finds max amplitude in each
        maxValzA= zeros(1,numLEDs);
        for m= 1:1:numLEDs
            maxValzA(m)= max(OS1(m),OS2(m));
        end
        maxValzA= maxValzA*100;


        maxValzB= zeros(1,numLEDs);
        for m= 1:1:numLEDs
            maxValzB(m)= max(bOS1(m),bOS2(m));
        end
        maxValzB= maxValzB*100;

% finds the difference between all the maximum
values and finds the greatest diffence
        diff= abs(maxValzB-maxValzA);
        keyToFind= max(diff)

% searched for bin containing the maximum
difference
        if keyToFind > sensitivity
            for find= 1:1:length(diff)
                if diff(find) == keyToFind
                    keys(x)= find;
                end
            end
        else
            keys(x)= 0;
        end
```

Code Snippet 4: Finding the onset of a note

4. LED SEQUENCING

4.1 Amplitude (VU meter)

The amplitude representation o the light strip works like a VU meter would. The zero points for an amplitude of zero are located in the center of the strip, at pixel 30 and 31. The left channel of audio is represented by pixels 1 to 30 and the right channel is represented by pixels 31 to 60. The matrix passed from the amplitude processing function is taken into the function and converted to commands for the Arduino to send to the LEDs. This conversion involves two formulas:

```
left= ((numLEDs/2)+1)-L;
right= ((numLEDs/2))+R;
```

Code Snippet 5: Calculating the left and right bounds of the display

these two formulas translate the amount of lights to be turned on each frame into a representation of that number as a distance from the center of the strip. For example if 10 lights are to be illuminated, than the formula will return lights 21 or 40 depending on which channel will be calculating. The command to be sent to the ardion then sends a range of the number returned by the left formula to the number sent by the right formula. That is, if the left magnitude is 10 and the right magnitude is 10, then the command will tell pixels 21-40 to illuminate. This is repeated for every frame. The commands are sent at a rate of 1/bins per second to keep in time with the music that is triggered at the beginning of the function.

```
pause(1/binsPerSec);
```

Code Snippet 6: this pause function ensures the commands are sent at the same rate of play of the audio

4.1.1. Bin Scaling

A special feature of this program is the ability to implement bins scaling. For audio files that are rather quiet, the amplitude integers sent to the sequencer may not exceed 5 or 6. This looks rather unimpressive; people want to see bright lights. This function scales that output of the analysis function to fill up as much of the strip as desired by the user by entering a percentage. If bin scaling is active and the percentage is set to 100, then all 30 pixels will be utilized for each channel. The maximum amplitude present in the original set of bins will be scaled to 30 and every number under it will be scaled to. The same process will work with any other percentage.

Original Matrix				Scaled Matrix			
3	3	2	2	18	18	12	12
2	2	3	3	12	12	18	18
2	2	2	2	12	12	12	12
3	3	2	2	18	18	12	12
2	2	3	3	12	12	18	18
2	2	3	3	12	12	18	18
3	3	2	2	18	18	12	12
3	3	1	1	18	18	6	6
2	2	5	5	12	12	30	30
2	2	3	3	12	12	18	18
2	2	3	3	12	12	18	18
2	2	3	3	12	12	18	18
3	3	3	3	18	18	18	18
3	3	5	5	18	18	30	30
2	2	1	1	12	12	6	6
3	3	0	0	18	18	0	0
3	3	0	0	18	18	0	0
2	2	0	0	12	12	0	0
2	2	0	0	12	12	0	0
2	2	0	0	12	12	0	0

Figure 2: The original results of the amplitude analysis versus the 100% scaling of the bins

4.2 Spectrum Display

The spectrum display work with all sixty pixels representing the spectrum as discussed in section 3.2. the first process of this section is to take the maximum amplitude of each bin for each channel. The data sent represents both channels, the maximum of the channels will be displayed of each frame. The numbers 1 to 8 that are present in each bin of each frame correspond to the intensity of the frequencies in that bin. This, the numbers will coordinate with a color. This allows the LEDs to not only represent which frequencies are present, but the intensity of that frequency as it exists in its bin. The frames are played back at the same rate as the music which is triggered at the beginning of the function. This rate is equal to 1/ bins per second.

```
colors= ['f','m','g','b','c','y','o','r'];
preparedBins(i,j)= max(bins(i,1,j),bins(i,2,j));
```

Code Snippet 7: The array of color keys corresponding to the intensity of a specific bin. The process of taking the maximum intensity of the left and right channels for each bin of each frame

4.3 Onset Detection Display

In the onset detection display. data passed to this function fills an array that, for each frame, has 1 or zero in each of the sixty bins. A 1 in a bin corresponds to an onset of a note being detected. For each frame a pixel will illuminate in a bin where the onset is detected. More than one pixel may illuminate if multiple onsets are detected.

```
if keys(i) ~= 0
        writeColor(nstrip, keys(i), 'r');
else
       writeColor(nstrip, 'f');
end
```

Coode Snippet 8: The pixel will illuminate if the bin contains a 1

5. CONCLUSION

While not the most efficient or consumer friendly audio visual application available, the program developed here has provided a valuable opportunity for us to investigate, research, and apply the audio processing techniques we've learned throughout the semester.

In addition, we hope we have contributed to the limited body of work available to DIY'ers interested in MATLAB to Arduino interfacing for musical LED control. The superior analysis tools of MATLAB were very helpful for processing the audio, and they offer much more functionality than provided by the Arduino IDE alone.

Our bin scaling algorithm in particular should be helpful for anyone attempting to display audio amplitude though LEDs while utilizing the full LED strip, regardless of actual song amplitude.

7. REFERENCES

[1] Frigo, M., and S. G. Johnson. "FFTW: An Adaptive Software Architecture for the FFT." Proceedings of the International Conference on Acoustics, Speech, and Signal Processing. Vol. 3, 1998, pp. 1381-1384.

[2] Tim Youndblood , Arduino Interface With MATLAB, All about Circuits, June 15, 2015

[3] Phillip Burgess, "Adafruit NeoPixel Uberguide", October 12, 2017