H2wOah

David Kunstmann, Kyle Ohlschlager, Madhu Ashok, Erik Nunez

University of Rochester

ABSTRACT

H2wOah is a chromagram representation of music and audio that allows for viewers to have an intuitive yet interactive experience. This device also lets non-musicians learn about basic tonalities as they appear in front of them. The approach used was determined to be successful; as the project does react in real time to the music that is used as input, but also that electrically "noisy" environments, or audibly noisy environments, depending on the input signal source (audio file versus microphone input), can affect the accuracy of the chromagram.

***Index Terms*—** Chromagram, DMX, Arduino, 12C

1. INTRODUCTION

The goal of this project was to implement communication between Arduino, MATLAB, Digital Multiplex 512 (DMX or DMX512), and motor control, ultimately displaying a chromagram of a processed audio track on a fountain of 12 motors for each half step in western music. This was solved using shields for Arduino, one of which developed by Adafruit for 8-bit (0-255) motor/pump control. Our group decided to use Arduino processors for the simplicity of loading new files, since the fountain is still in the prototyping stage of development. DMX protocol was used for controlling an LED illumination unit with 12 mappable RGB lights impinging the water jets. Arduino allows for communication at 9600 bits per second, and served as a communication link between the software processing the audio content in real time. Our algorithm mapped parameters to 8-bit resolution, which communicated on the 12C bus, native to each Arduino Uno unit and helped with debugging issues with boards.

2. BACKGROUND AND EXPERIMENT

2.1. DMX Protocol

Digital Multiplex 512 is a way of communicating between lighting units that are connecting in a daisy chain. There are up to 512 channels of control, with variables mappings from 0-255. DMX 512 can be coupled with Arduino through a shield by Conceptinetics, exponentiating control.

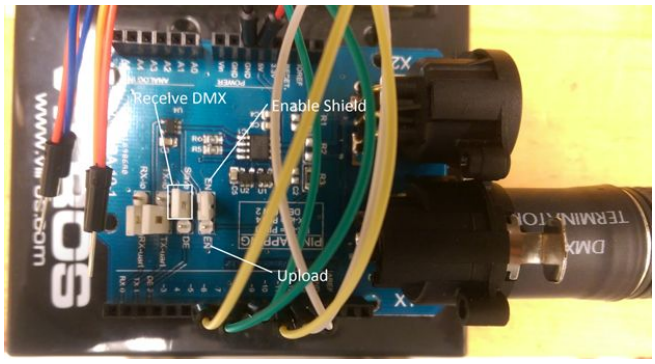


Figure 1: Conceptinetics shield for Arduino with various jumper controls.

2.2. Arduino Shields

Arduino microprocessors are used for their availability of open source projects, and compatibility with varying voltages/signals in prototyping experiments. Four processors were used to reduce noise in the signal. A 'Master' Arduino is used for communicating with MATLAB values of the frequency content, three 'Worker' Arduinos are used to receive 8-bit code and pump the fountains with a burst. Two DMX shields were used. one to transmit signal to the lights, and the other to receive DMX input from an input. Motor shields were used to control speeds of the fountain from 0-255. A motor shield can stack with a DMX shields:

Figure 2: Stacking DMX and Motor Shields on Arduino.

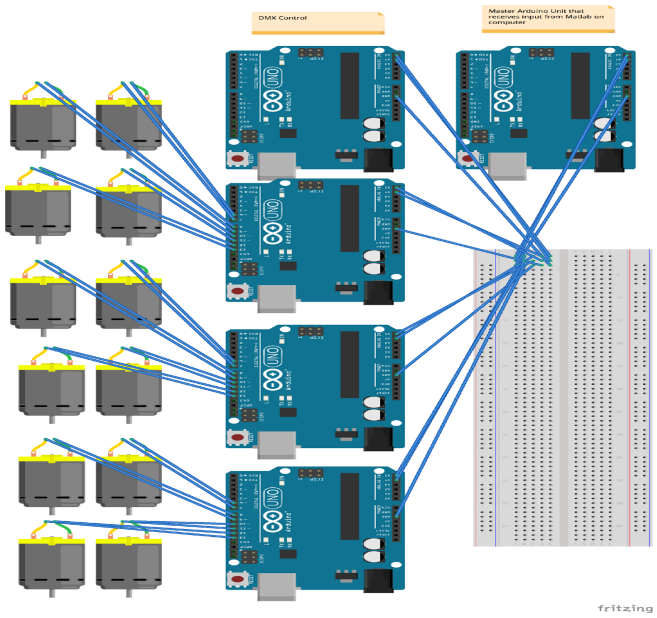


Figure 3: Concept Diagram of Arduinos, Water Pumps and DMX Shield.

2.3. Chromagram

We implemented a chromagram algorithm in homework #5 but for this fountain we have used a more robust algorithm which uses midi note pitch classes to identify scale number from a real-time spectrogram. Below we can see the so called conversion matrix which is used to multiply each incoming frame of the power spectrum of the signal by a matrix of zeros and ones so as to map frequency bins from the spectrogram to 12 notes of one octave.

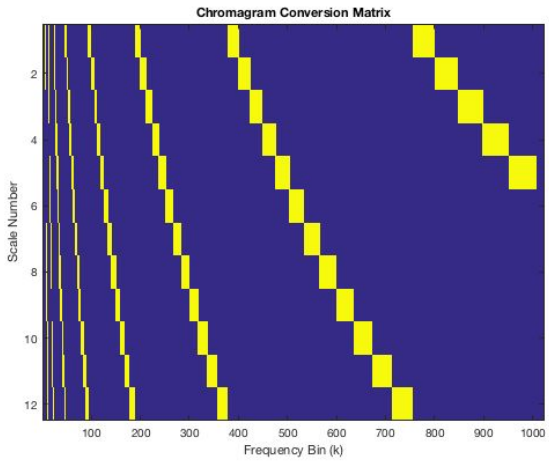


Figure 4: Chromagram Conversion Matrix

Additionally, we had to modify our algorithm to operate using the Real-Time Toolbox in matlab. This way we were able to process and send audio data from matlab to the water pumps and DMX controlled light in time with the music or input source. Here is the basic operation of the Matlab to Arduino serial port through which the lights and pumps communicate with the audio processing. This is only code showing the basic idea of writing to arduino using the given functions

```
%Declaring the arduino serial port
arduino = serial('/dev/cu.usbmodem1411',
'BaudRate', 9600);

%open serial port
fopen(arduino);

%audio processing/writing loop
tic;
while(toc < 10)
    %Do audio processing
    fwrite(audio)
end

fclose(arduino)
clear arduino %need to clear port before next
run
```

At first we could not entirely figure out how to process and send data out of Matlab via a serial port to Arduino using the Real-Time Toolbox. We tested out our algorithm instead only processing an audio file and not being able to hear the music properly at the same time. Using this test file we generated a chromagram which be able to print the data properly but not send it. Here is a chromagram processed frame by frame but the audio did not play during the processing loop and the image below was displayed after the loop the loop ended.

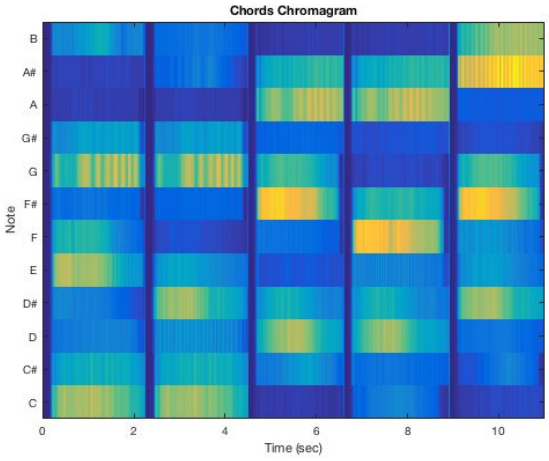


Figure 5: Concept Diagram of Arduinos, Water Pumps and DMX Shie

The scaling or normalizing of the chromagram data from power per note of the octave as seen above, to values from 1 to 255 for DMX and from about 75 to 255 for the water pumps was done using a separately written adaptive

normalize function. The DMX values correspond to RGB values for each of the 12 lights and motor speeds or motor speed cutoffs for the water pumps. The values were scaled from power to 12 values in the ranges specified above. One scaling for the pumps and one for the lights.

```
 if (chrome[6]>150){
   motor7->setSpeed(255);
   delay(50);}
 else if (chrome[6]<150) {
   motor7->setSpeed(0);  }

 if (chrome[7]>150){
   motor8->setSpeed(255);
   delay(50);}
 else if (chrome[7]<150) {
   motor8->setSpeed(0); }

 // Channel 1
 if (chrome[0]<100){
   dmx_master.setChannelValue ( 1, chrome[0] ); }
//Red 1
 if (100<chrome[0]<200){
   dmx_master.setChannelValue ( 2, chrome[0] ); }
//Green 1
 if (chrome[0]>200){
   dmx_master.setChannelValue ( 3, chrome[0] ); }
//Blue 1
 if (chrome[0]<5){
   dmx_master.setChannelValue ( 1 , 0);
   dmx_master.setChannelValue ( 2 , 0);
   dmx_master.setChannelValue ( 3 , 0);  }
```

Figure 6: Code excerpt from an Arduino sketch for 'Worker' controlling motor speeds and lights simultaneously.

2.4.Troubleshooting

The DMX and motor shields had issues interfacing with stacking, due to shields sharing ports of the Arduino. Additionally, the units needed to be reset often to clear the RAM of the microprocessors. Each pump required around .4 Amps to perform ideally.

3. CODE

Various programs were used in conjunction with MATLAB, which we chose to be central to the audio signal processing. A communication bus at 9600 bits per second is initiated:

```
arduino = serial('COM9', 'BaudRate', 9600); %
create serial communication object

fopen(arduino); % initiate arduino
communication
```

Figure 7: Code excerpt from MATLAB. 'COM9' references the USB port.

Bits of information are sent from MATLAB to the Arduino, which is received on the 'Master' channel. The processor checks for available data on the serial port, and stores the 8-bit integers in a [1x12] array.

```
 if(Serial.available()) {
   for(int i = 0; i < 12; i++) {
     temp = Serial.read();

     if(temp < 165) {
       chrome[i] = 0;
     }
     else {
       chrome[i] = temp;
     }
     Serial.print(chrome[i]);
   }
 }

 Wire.beginTransmission(11);

 for(int i = 0; i < 12; i++) {
   Wire.write(chrome[i]);
 }
 Wire.endTransmission();
```

Figure 8: Code for 'Master' Arduino. sending chromagram values for each of the 12 channels.

The 'Master' channel then sends the chrome[i] array to other Arduinos on the 12C bus. The transmission channel will be #11:

```
void setup() {

  // MOTOR SHIELD
  AFMS.begin(20);     // 20Hz chosen for cutoff frequency
of pumps

  motor5->run(BACKWARD);
  motor6->run(BACKWARD);
  motor7->run(BACKWARD);
  motor8->run(BACKWARD);

  Wire.begin(11);
  Wire.onReceive(receiveEvent);

  // Enable DMX master interface and start transmitting
```

```
dmx_master.enable ();

  // Set channel 1 - 50 @ 50%
  //dmx_master.setChannelRange ( 1, 25, 127 );
}
void receiveEvent(int bytes) {
  for(int i = 0; i<12; i++){
    chrome[i] = Wire.read();
    //chrome[i] = chrome[i];
  }
}
```

Figure 9: Code for 'Worker' Arduino receiving 8-bit integers using Wire.read().

The communication link with 12C allows for data transfer among arduinos and MATLAB.

4. PROTOTYPING

The chromagram fountain can operate for discrete intervals of audio (entire song lengths) when clean power is available to each of the 12 water pumps. A computer is necessary for operation, but only one USB port is required to communicate with MATLAB. The device can read DMX signal through one of the 'Worker' Arduino boards. This was helpful during the troubleshooting phase. Shields, power cords, wires, and processors were swapped to test every component of the project.

```
fftlength = 1024;
hopsize = 512;
framesize = 1024;

Fs = 44100;

% create serial communication object
arduino = serial('COM9', 'BaudRate', 9600);

fopen(arduino); % initiate arduino
communication

t = 0:1/Fs:10;
s = sin(2*pi*440.*t);

%'04 Wildcat.mp3'
fileIn = dsp.AudioFileReader('04 Wildcat.mp3',
'SamplesPerFrame', framesize);
audioIn = audioDeviceReader;
audioOut = audioDeviceWriter;

filename = 'convM.mat';

C = conversionMatrix(fftlength, 44100);
save('convM.mat', 'C');
```

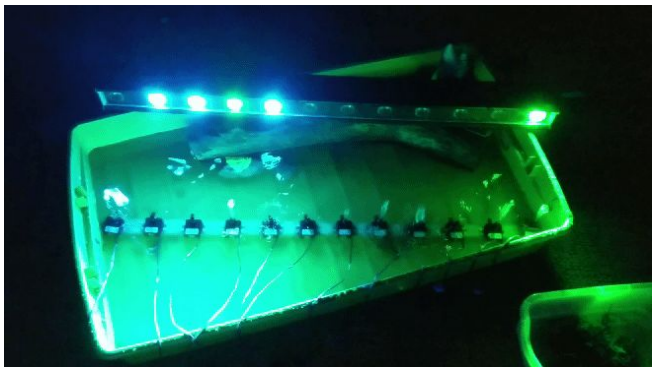
```
%seconds to run real-time loop
time = 10;]

k = 1;
hamm = hamming(framesize);
min = 240000;
tic;
while(toc < time)
        audio = step(fileIn);
        audio = 0.5.*audio(:, 1);
        frame = fft(audio.*hamm, 2*fftlength);
        freqHz =
(Fs/(2*pi)).*frame(1:length(frame)/2);
        frame2freq(:, k) = real(abs(freqHz));
        framemag2db(:, k) =
mag2db(abs(freqHz));
        chrome = C*frame2freq(:, k);

        audioOut.step(audio);

        chrome = normalize(chrome, 1, 255);

        fwrite(arduino, chrome, 'uint8');
        disp(chrome);
        k = k + 1;
end

fclose(arduino);
clear arduino;

release(audioOut);
release(audioIn);
release(fileIn);
```

Figure 10: Chromagram implementation in MATLAB, which communicates with Arduino on 12C.

This converted the 'Worker' arduino into a DMX controller, and MATLAB into DMX compatible software.

5. CONCLUSION

In conclusion we have realized a way of representing music in real time via a water pump and lighting system. Furthermore this system is controlled via Arduino and receives real time data from Matlab. In the future we hope to isolate note of the scale in a more robust way and stabilize the arduino system in terms of more reliable power supplies and delay matching of the loop functions between each arduino and the master arduino controller. Some future ideas for applying this concept are integrated structures in public places which respond to live input and run on solar power. Explore other concepts of Chromesthesia with artists who animate and create music based on what they draw.



6. REFERENCES

[1] “Adafruit Library Reference”. Adafruit Motor Shield V2 for Arduino(2017) <https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/library-reference>

[2] “Chromagram Representation for Music Signals”. (2017) <http://www.ece.rochester.edu/~zduan/teaching/ece472/lectures/Chromagram.pdf>

[3] “DMX Library for Arduino”. Conceptinetics.(2017) <http://sourceforge.net/projects/dmxlibraryforar/files/>