Audio Pitch Estimator with Frequency & Sound Reactive Lights

D.M.P.M. Alwis (E/16/019), P.D.R. Fernando (E/16/103),
I.M. Insaf (E/16/142), I.Z.M Zumri (E/16/415)

12th December, 2021

Introduction

The five senses each excite a separate part of your brain. Looking a bright colour stimulates the primary visual cortex. A person with synesthesia might also get a taste of what they see right while they look at it. The same applies to sound. A person with synesthesia is believed to have coloured hearing. That is they see sounds, music or voices seen as colors. This is a beautiful syndrome. With our product, we intend to give a feel of this to ordinary people.

Sound has many fundamental elements. Of all the properties of sound, pitch, tonal quality and gain can be deemed to play a major role in the quality of music. A frequency's sensation is generally referred to as a sound's pitch. Pitch is the foundation of melody and harmony. Tonal quality refers to the richness or perfection of musical tones. Gain is the loudness or the power in the music. A visual representation of a combination of all these makes it a lot more interesting. The product we designed does just this, interrelating all these features of sound to give the user a feel of the interesting syndrome of synaesthesia.

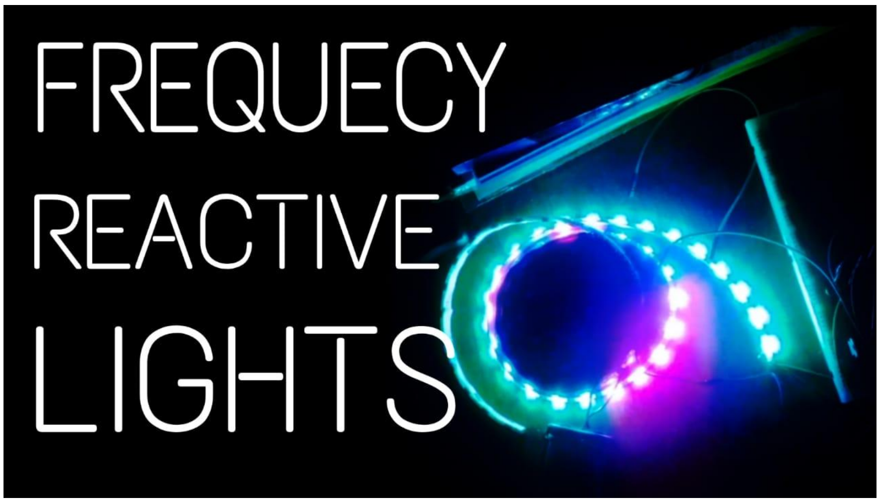

Figure 1: A pattern of light in the LED strip

Market

LED strip lights connected to pitch estimators have become popular among the youth these days especially among the musicians, gamers and Youtubers. They have been used as a means to express the feeling, to create a nostalgic environment and to bring more color to what they do.

Furthermore, a user can elevate what he/she feels while listening to music when the environment is set to react to music.This will psychologically affect the user in a much more positive way.

This can be used in many products to elevate their product.For example the below figure contains products that can be elevated using frequency reactive lights.Furthermore, clubs,hotels,pubs can be decorated to improve the user experience.

Figure 2: Decorative Lights

The cost analysis that we performed on the product is shown below.

Component	Cost /Rs.
Microcontoller-ESP8266 / ESP32 /PIC12F683	100.00 to 400.00
PCB	400.00
Microphone	20.00
LM386	20.00
5V regulators, capacitors, resistors, potentiometers, wires	120.00
Light	880.00
Power pack	500.00
Casing	500.00

Total	**2540.00 to 2840.00**
Selling Price	**9990.00**
Gross Profit	**7450.00 to 7150.00**

The profit margin was decided by studying similar products in the market. These products which have much less features compared to our product sell for around Rs. 6000. This made us decide the price tag on our tag is very reasonable and that people would be willing to spend a little more for the extra features that we offer.
We intend to market the product to musicians, social media influencers, gamers, pubs and hotels.

Methodology

The input is obtained using the microphone and processed in the microprocessor. Fast Fourier Transform (FFT) was initially used to analyze the frequency spectrum of the signal. FFT is quite accurate but there arises issues when it comes to choosing the window size. A large window size estimates the amplitudes of both higher and lower frequency components accurately. But a large window size means a higher number of samples which results in larger processing time. This is a problem because the device needs to give out the visual representation in real time.Furthermore larger window size may not estimate the higher frequencies such as symbols and hi-hats properly since higher frequencies tend to decay faster than the lower frequencies which results in low amplitudes for higher frequency signals as the averaged value through out the window is low. A smaller window size has a lower resolution for frequencies. But it detects the amplitude of higher frequencies with good accuracy. A tradeoff between resolution and process time needs to be considered when selecting the window size. As a result, we used the Goertzel Algorithm to estimate the pitch.

In the frequency domain, notes are discreetly spread out throughout the spectrum. Therefore, pitch can be obtained by finding the frequency (music note) with the highest power in the spectrum of the input signal. Goertzel algorithm was used to obtain the power spectral density values at each note or frequency. In our program, frequencies of the notes were predefined. Then, they were normalized using the sampling frequency. For each predefined frequency, a digital Goertzel filter was designed. The filter equation in the time domain is given below.

$$q[n] = x[n] + 2 \times cos(\omega) \times q[n - 1] - q[n - 2]$$

$$y[n] = q[n] - e^{-j\omega} \times q[n-1]$$

where,

ω *is the normalized frequency of the desired note*

$q[n]$ *is an intermediate signal*

$y[n]$ *is the filter output signal*

Here, the transfer function of the Goertzel digital filter is given by,

$$G(z) = \frac{1}{1 - e^{+j\omega} \cdot z^{-1}}$$

where,

ω *is the normalized frequency of the desired note*

From these equations, an equation to obtain the power spectral value at the desired frequency was derived.

$$Power\ at\ \omega = Y[k].Y^{*}[k] = \sqrt{q[n-1]^2 + q[n-2]^2 - 2 \cdot cos(\omega) \cdot q[n-1] \cdot q[n-2}$$

where,

ω *is the normalized frequency of the desired note*

Using this equation, the power level was calculated. The power level at each predefined frequency for each sample was then normalized. Then the average power level at each desired frequency was calculated using a cumulative moving average algorithm.

$$P_{avg} = a \cdot P_{now} + (1 - a) \cdot P_{previous,\, avg}$$

where,

a *is constant between* 0 *and* 1

Finally, the frequency with the highest power was taken as the pitch of the audio signal. These power values are updating in real time. Therefore, we get the instantaneous pitch of the input signal.

The ESP32 microprocessor was used initially. This was later replaced by the ESP8266 microprocessor since the FFT algorithm was ditched and a higher clock speed no longer required, which is a cost effective option. ESP8266 has only one analog pin but we have 4 analog parameters to handle(Colour,LEDs,Modes,Mic Input). A 1-bit MUX was used to read both the microphone input and the input from the user parameters. This is done every loop. Push buttons were used so that the user could specify which feature is being varied by the knob.

The microphone input was affected largely by electrical noise. To overcome this, we used an amplifier to amplify the microphone input. The unwanted frequencies were filtered off using a combination of resistors and capacitors.

The frequencies are represented in an LED strip. The Scriabin Circle was used to select the colours. This relates to a rare medical condition called Synesthesia in which the stimulation of one sense causes the automatic response of another sense. Researchers believe that the most common form of synesthesia is colored hearing: sounds, music or voices seen as colors. The Scriabin circle is also developed using the same concept.The reason we used this concept to choose colours was to give the users an idea of Synaesthesia.

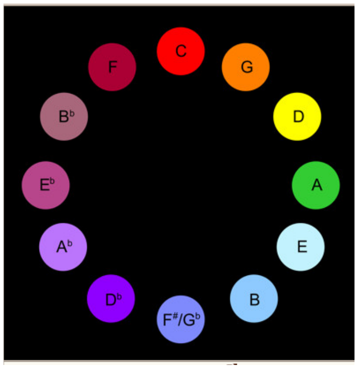

Figure 3: The Scriabin circle: *Adapted from* https://www.flutopedia.com/sound_color.htm

When the light is reacting just to the energy level of the song in order to display the best colour combinations, Colour Theory was used. Colour Theory is based on the use of a color wheel to guide color mixing and the visual effect of a certain color combination. Red, yellow, and blue were the primary colors of the classic color wheel, known as the colour circle. The colour circle has evolved into the colour wheel over time. This has enabled it to display a wider range of colours and also utilize the primary, secondary and tertiary hues. The following colour schemes were used;

- Complementary
- Split complementary
- Tertradiac
- Triadic
- Analogous

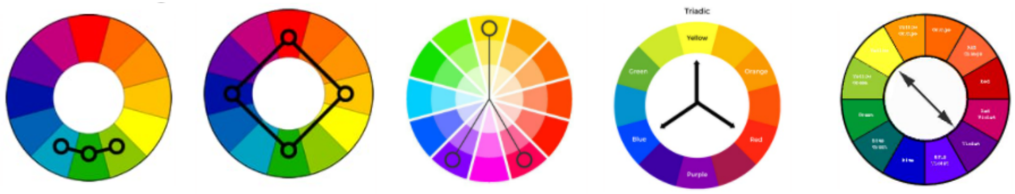

Figure 4: The five different Colour Schemes

Schematic and PCB

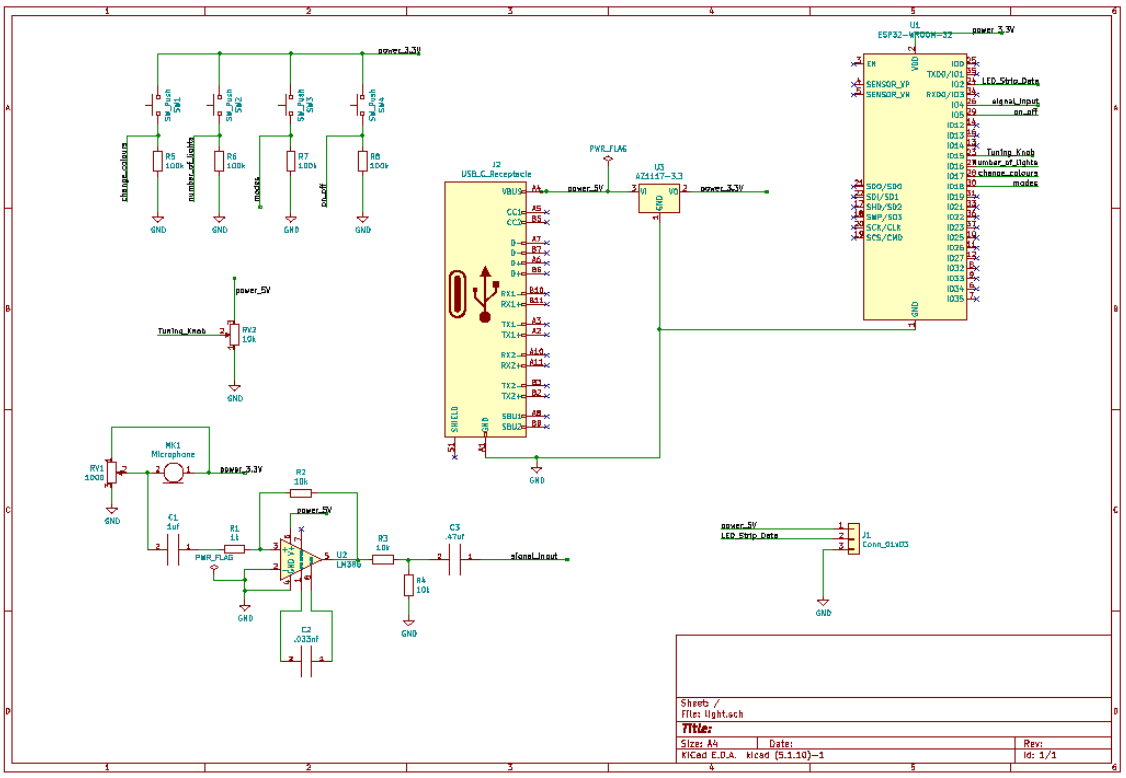

Figure 5: Schematic of the circuit

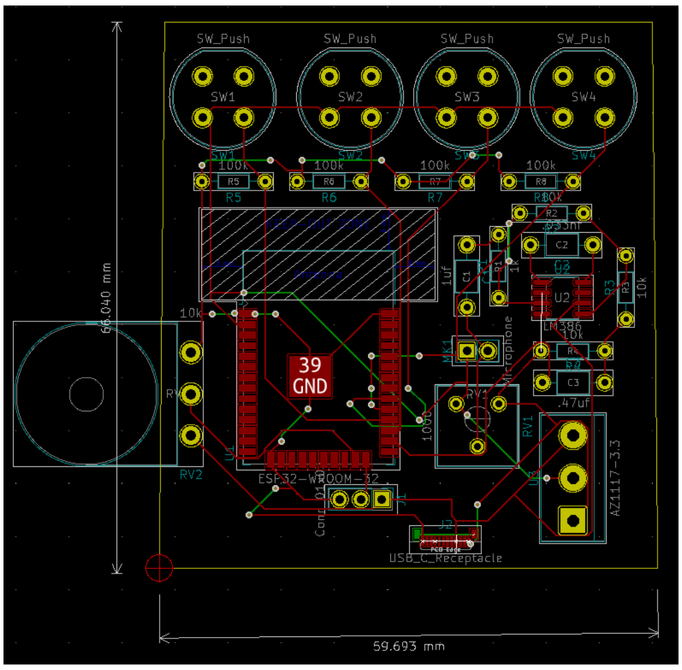

Figure 6: Printed Circuit Board

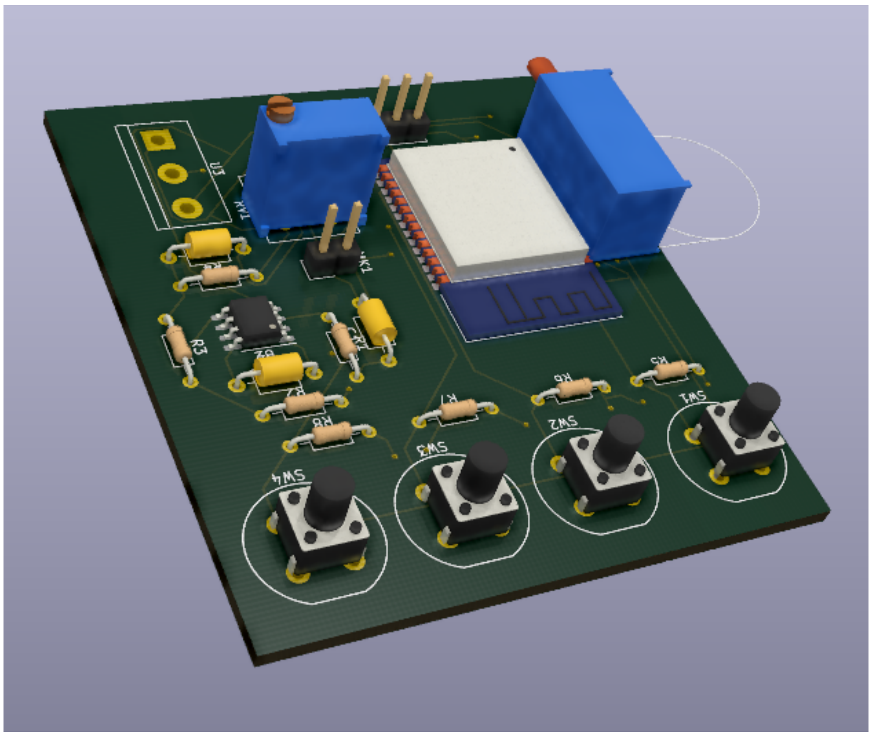

Figure 7: 3D CAD file of the PCB

Design

The design was done keeping ergonomics in mind. Simplicity and ease of use were our main focus. The device consists of a rotating knob, four oval shaped buttons, grid for the microphone and input/output ports. The knob sits right in the middle of the device making it the main part of the design.

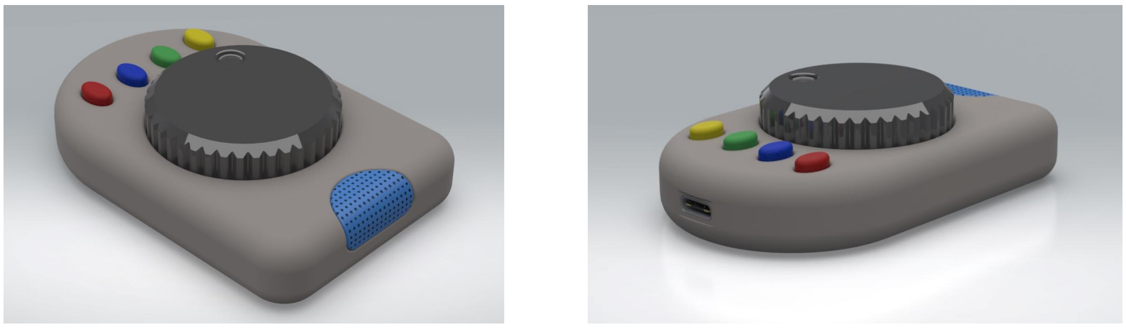

Figure 8: Isometric View

The buttons are chosen in four different colors making them easily distinguishable. It also symbolizes the color lighting used with this device. The knob and the buttons are large enough so that they can be easily handled even in the dark.Colors were chosen to give the device a classy and elegant look.Overall device is small enough making it visually pleasing in a decorative environment.

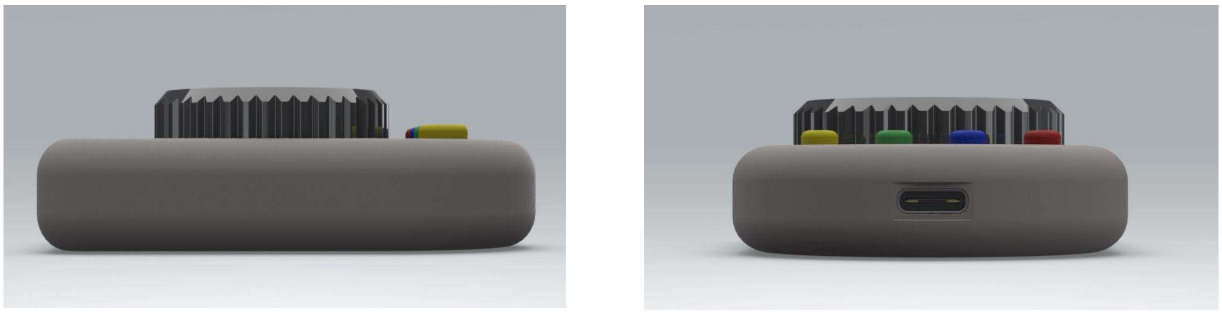

Figure 9: Side and Front View

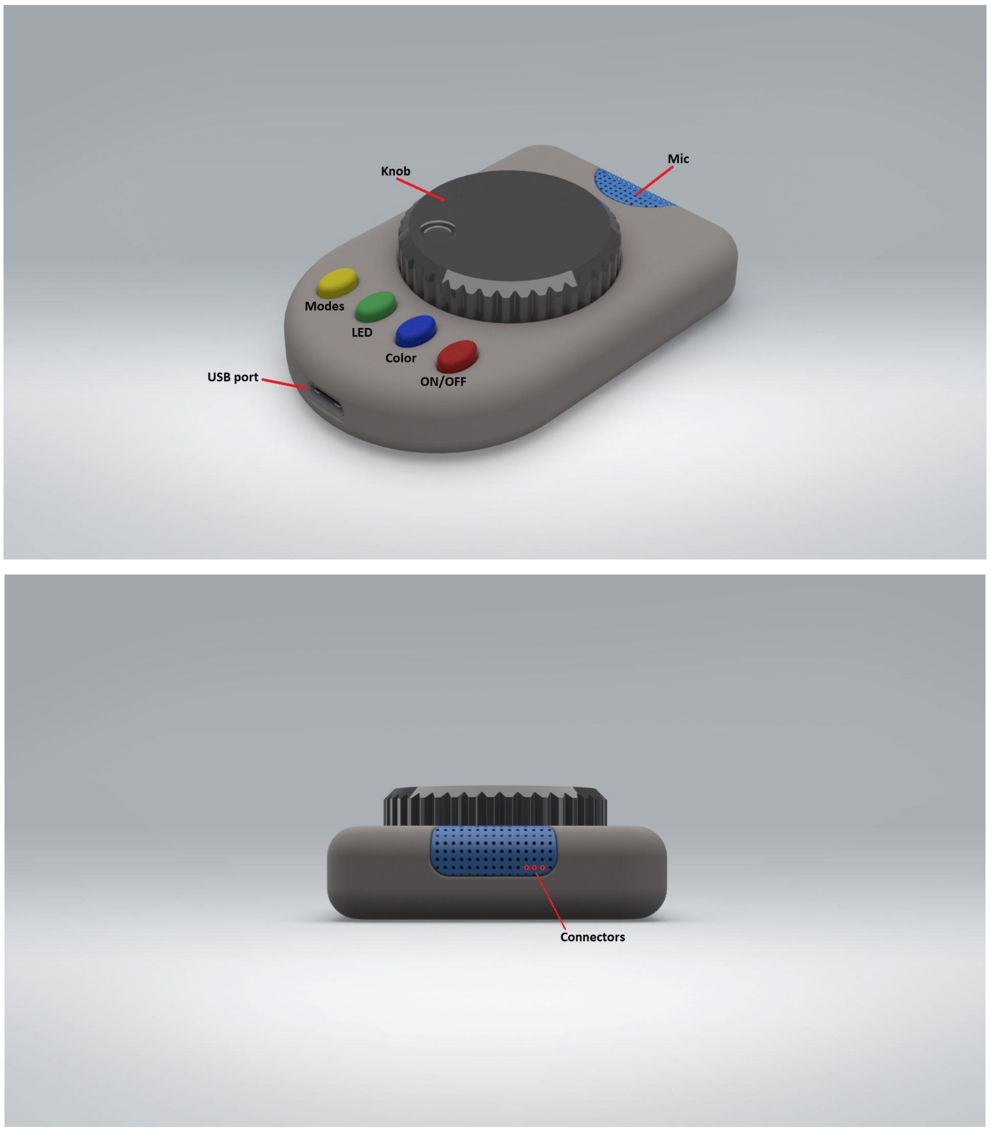

Figure 10: Components of the device

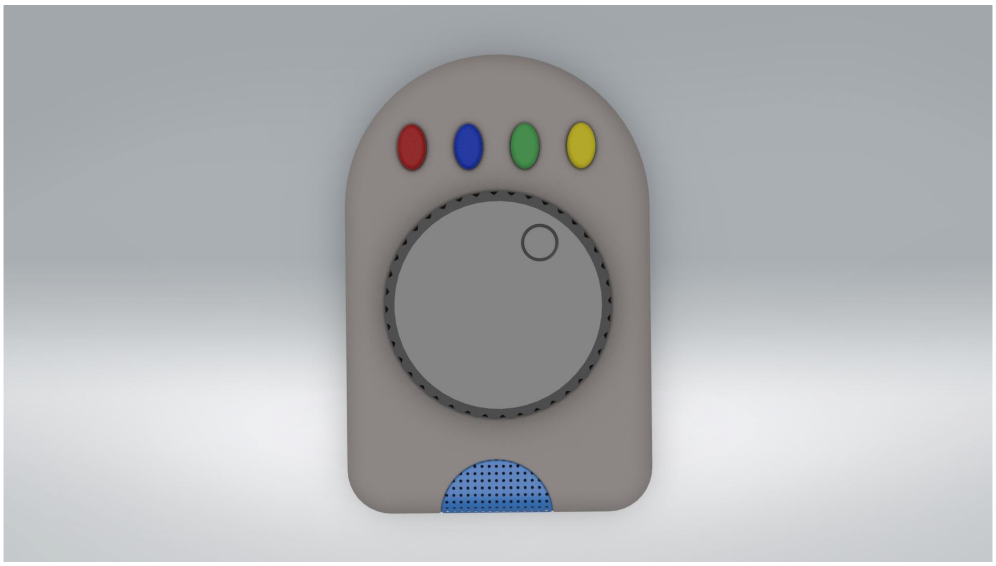
Figure 11: Top View

How To Use?

- Place the LED strip according to the preference.(Walls,Behind the TV...etc).
- Connect the LED strip to the controller via 3 male header pins.(Make sure to place the controller near the music source.)
- Plug the power adopter through the USB port.
- Switch on and change the parameters accordingly.
- Enjoy the show

Functionality of the components

Knob : Used to change the parameters; Color, Modes, LED

Buttons :

ON/OFF	Power on and off the device
Color	press the color button to select the variable parameter as colour (hue angle) Then rotate the knob to obtain the prefered colour. (By rotating the knob hue angle is changed which is a parameter how color is defined)

LEDs	If the product cannot function as the existing market product which is the LED strip (Not reactive to sound or frequency) is a drawback. Hence this button is used to select which portion of lights are lit in the LED strip and which portion is reacting to sound/frequency.This way the user has the flexibility to adjust the dynamics of the light.
Modes	There are 29 modes the user can enjoy which includes lighting patterns to color settings (Complementary,Split complementary,Tertradiac...etc)
USB port	To provide power to the circuit and lights.USB port is used because USB charger is something that is readily available in most of the modern homes,personals...etc.This way the cost is reduced for the product since it is a customer choice to request a usb charger. Alternative for this case is using a DC power barrel jack to obtain power.
Connectors	Connect the LED strip with the device.

Future Developments

- One of the main developments that we intend to work on is to detect the genre or the mood of the music using neural networks. This would enable a more realistic visual representation of the music. This is done by using different colours to represent different moods. The base work for this such as finding the spectral centroid has already been carried out. Below is a plot made by considering the area of spectra of a few songs of different genres. We can see that it is a bit of a complex dataset to cluster.
  (Matlab code can be found in the appendix)

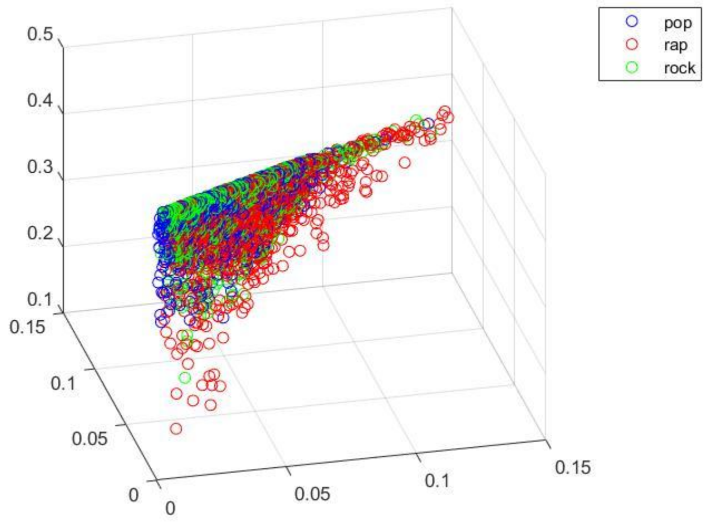

Figure 12: 3D feature plot of spectrum area for different songs of different genres

- The current design requires the device to be placed close to the source of music. We intend to use more effective methods to ensure music and noise separation so that the device could ignore sounds which aren't musical.

- Develop it to the extent that music producers could embed the colour pattern files into the music as they produce. When music is distributed around the world, users can enjoy the song's visual interpretation of the musician through their eyes.

- Develop a mobile application so that the phone's microphone could be used as the microphone that picks up the music. This would reduce the cost further as it overcomes the need of an additional microphone. This method could result in a delay of what is being displayed due to the delays in data transfer between the phone and our device. Furthermore a main disadvantage in this design is that users have to physically contact in order operate the device.Mobile application can give a solution to this problem too.

- Connect the device directly to musical instruments and display the light patterns using hexagon LED lights.This way the product can be enjoyed remotely.
  (ex : Camping)

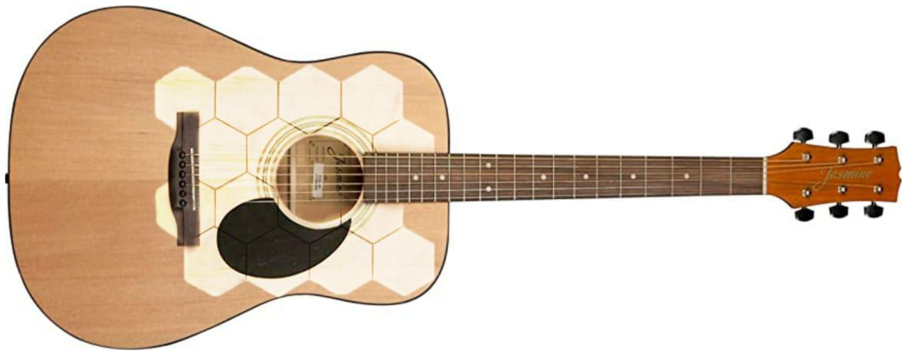

Figure 13: A guitar with hexagonal lights

Problems Faced

- The main problem faced is, for the FFT algorithm, a fixed window should be predefined in order to obtain the magnitude spectrum. Until this window is processed, no input is buffered. So, in order to maintain a consistent sampling, the sampling frequency was adjusted such that processing is done without distortion. This reduced the sampling frequency drastically.

  Two solutions were tested to overcome this problem. The first option was to replace the microcontroller with a microcontroller with very high clock speed. This will nimble the FFT processing which will indirectly increase the sampling frequency.

The main drawback of this solution was the cost. Microcontrollers with very high clock speed are very expensive.(ESP 32 - 240MHz)

Furthermore, the FFT algorithm gives the complete magnitude spectrum which executes a higher number of operations. The microcontroller consumes a higher number of clock cycles and hence the sampling time should be higher to obtain the desired result.

For pitch identification, what's important is identifying the frequency with the highest power. The frequency with the highest power will give the note or the pitch of the audio signal under low noise. Moreover, the number of music notes is finite. So if we take the power spectral density values at these frequencies, the dominant frequency or the pitch can be identified. One suggestion was to filter out the noise and calculate the power spectral density at each frequency (note). This option was chosen since this is the most cost effective option since we can advance to a microcontroller with lower computational power because It gets the job done.

- The z transform of the Goertzel filter used is given by,

$$G(z) = \frac{z}{z - e^{+j\omega}}$$

This filter is marginally stable since there is a pole on the unit circle. This can become unstable due to computational inaccuracies associated with floating point arithmetic. This can cause variables to overflow.
In order to overcome this, the power values obtained at each sample are first normalized by dividing them using the maximum power value and then these normalized values were averaged using cumulative moving average algorithm. Here gaussian normalization was not used since it contains negative values which tends to give erroneous results for average power.

FAQ

1. How long is the LED strip?

   Currently ws2812b LED strips are available in the market in lengths of 1m/2m/3m/5m/10m

2. How do you power up the device ?
   Use a regular USB type-C cable with a power adapter or power bank.

3. Can I use this device to tune music instruments ?
   No, the purpose of this device is to give a visual representation of the sound and is not used for tuning.

4. Can this be used with different LED strips?
   No, it is only compatible with ws2812b LED strips.

5. Should the device be kept near the musical instrument?
   Yes, we recommend placing it near the musical instrument for an accurate visual representation free from noise and other disturbances.

REFERENCES

https://www.colormatters.com/color-and-design/basic-color-theory

https://www.flutopedia.com/sound_color.htm

mtu.ehttps://pages.mtu.edu/~suits/notefreqs.htmldu

https://www.circuitbasics.com/build-a-great-sounding-audio-amplifier-with-bass-boost-from-the-lm386/

https://www.allaboutcircuits.com/technical-articles/introduction-to-usb-type-c-which-pins-power-delivery-data-transfer/

https://color.adobe.com/create

https://www.reddit.com/r/arduino/comments/gfuzdx/music_visualizer_audio_normalization/

https://en.wikipedia.org/wiki/Fast_Fourier_transform

APPENDIX

Main Code With Goertzel Algorithm

```
//freq range - 150 Hz to 4300 Hz

#include "arduinoFFT.h"
#include "FastLED.h"
#include "Arduino.h"

arduinoFFT FFT = arduinoFFT();

#define CHANNEL 4
#define DIGITAL 5

#define NUM_LEDS 60

#define DATA_PIN 2
#define CLOCK_PIN 13

//#define Hue_pot (data pin of knob to change hue)                    //edit this part when interrupts are available and pots
//#define baseLED_pot (data pin of knob to change no of baseleds)

const uint16_t samples = 128; //This value MUST ALWAYS be a power of 2
const double samplingFrequency = 9000; //Hz, must be less than 10000 due to ADC

unsigned int sampling_period_us;
unsigned long microseconds;

// Define the array of leds
CRGB leds[NUM_LEDS];

/*
These are the input and output vectors
Input vectors receive computed results from FFT
*/
double vReal[samples];
double vImag[samples];

#define SCL_INDEX 0x00
#define SCL_TIME 0x01
#define SCL_FREQUENCY 0x02
#define SCL_PLOT 0x03

#define pi 3.14159265359
#define ADCCENTER 512

double m;
double ave,val=0;
int count=0;
int pre_count = 0;
uint8_t hue;
uint8_t hue1[3];
int baseLed=20;
double Vibe = 1;   //to capture portion of energy of a frequency band.this should be analog Read
int red,green,blue =0 ;
float x;
int push_button=12;


// Variables for Goertzel algorithm
double FA = 880;
double FB = 987.77;
double FC = 523.25;
double FD = 587.33;
double FE = 659.25;
double FF = 698.46;
double FG = 783.99;

// Note names
char names[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G'};
// Power values
double psdMag[7] = {0, 0, 0, 0, 0, 0, 0};
double prob[7] = {0, 0, 0, 0, 0, 0, 0};


// Normalizing
double Fs = samplingFrequency;
double w[] = {(2 * pi * FA) / Fs, (2 * pi * FB) / Fs, (2 * pi * FC) / Fs, (2 * pi * FD) / Fs, (2 * pi * FE) / Fs, (2 * pi * FF) / Fs, (2 * pi * FG) / Fs};
double q0[7] = {0, 0, 0, 0, 0, 0, 0};
double q1[7] = {0, 0, 0, 0, 0, 0, 0};
double q2[7] = {0, 0, 0, 0, 0, 0, 0};
double sample = 0;


void setup(){
  Serial.begin(256000);
  pinMode(CHANNEL, INPUT);
  pinMode(DIGITAL,INPUT);
  sampling_period_us = round(1000000*(1.0/samplingFrequency));
 FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
 for(int i=0;i<50000;i++)
 {
   ave = ave+ analogRead(CHANNEL);
 }

 ave = ave/50000;
}





void loop()
{

//taking audio buffer
  microseconds = micros();
  for(int i=0; i<samples; i++)
    {
      sample = analogRead(CHANNEL);
      vReal[i] = sample - ave;
      vImag[i] = 0;

      // Goertzel Algorithm
      int note_ind = goertzel(sample);
      Serial.println(names[note_ind]);
      while(micros() - microseconds < sampling_period_us){
        //empty loop
      }
      microseconds += sampling_period_us;
      Serial.println(vReal[i]);
  }

 fourier();

  for(int i=0;i<round(Vibe*sizeof(vReal)/sizeof(double));i++)
   {
      m +=abs(vReal[i]);
   }

  count = round(map(m,0,100000,0,50));

  bassColor();

  selectmode(push_button);

  pre_count=count;
  m=0;
```

```
}


int goertzel(int sample)
{
  double maxx = 0;
  for(int i = 0; i < 7; i++){ //Modify loop termination later
    q0[i] = ((float)sample - ADCCENTER) + 2 * cos(w[i]) * q1[i] - q2[i];
    q2[i] = q1[i];
    q1[i] = q0[i];

    psdMag[i] += sqrt(q1[i]*q1[i] + q2[i]*q2[i] - 2 * cos(w[i]) * q1[i] * q2[i]);
    maxx = max(0, psdMag[i]);
    }

  int ind = -1;

  for(int i = 0; i < 7; i++){ //Modify loop termination later
    prob[i] = 0.7 * prob[i] + 0.3 * (psdMag[i] / maxx);

    if(prob[i] > maxx){
      maxx = prob[i];
      ind = i;
      }
    }
  return ind;
}



int selectmode(int var)
{
  switch (var)
  {
    case 1:
      complimentory();
      harmonyMode0();
      break;

    case 2:
      complimentory();
      harmonyMode1();
      break;

    case 3:
      complimentory();
      harmonyMode2();
      break;

    case 4:
      splitComplimentary();
      harmonyMode0();
      break;

    case 5:
      splitComplimentary();
      harmonyMode1();
      break;

    case 6:
      splitComplimentary();
      harmonyMode2();
      break;

    case 7:
      tertradic();
      harmonyMode0();
      break;

    case 8:
      tertradic();
      harmonyMode1();
      break;
```

```
case 9:
  tertradic();
  harmonyMode2();
  break;


case 10:
  analagous();
  harmonyMode0();
  break;

case 11:
  analagous();
  harmonyMode1();
  break;

case 12:
  analagous();
  harmonyMode2();
  break;

case 13:
  mode5();
  mode0();
  break;

case 14:
  mode5();
  mode1();
  break;

case 15:
  mode5();
  mode2();
  break;

case 16:
  mode6();
  mode0();
  break;

case 17:
  mode6();
  mode1();
  break;

case 18:
  mode6();
  mode2();
  break;

case 19:
  mode7();
  mode2();
  break;

case 20:
  mode7();
  mode0();
  break;

case 21:
  mode7();
  mode1();
  break;

case 22:
  mode7();
  mode2();
  break;

case 23:
  complimentory();
  mode3();
  break;
```

```
    case 24:
      mode5();
      mode3();
      break;

    case 25:
      mode6();
      mode3();
      break;

    case 26:
      mode7();
      mode3();
      break;

    case 27:
      splitComplimentary();
      mode3();
      break;

    case 28:
      tertradic();
      mode3();
      break;

    case 29:
      analagous();
      mode3();
      break;

  }
}




void fourier()
{
  //fft
  FFT.Windowing(vReal, samples, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
  FFT.Compute(vReal, vImag, samples, FFT_FORWARD); /* Compute FFT */
  FFT.ComplexToMagnitude(vReal, vImag, samples);
 //PrintVector(vReal, (samples >> 1), SCL_FREQUENCY);
  x = FFT.MajorPeak(vReal, samples, samplingFrequency);
}




void bassColor()
{
//  hue=analogRead(Hue_pot);
//  baseLed = analogRead(baseLED_pot);
  hue = 125;
  for(int i=0;i<baseLed;i++)
  {
    //leds[i] = CHSV(hue, 100, 100);
    leds[i].setHue(hue);
  }
}



void complimentory()
{

  if(hue<180)
  {
    hue1[0]=hue+127;
  }
  else
  {
    hue1[0]=hue-127;
  }

  hue1[1]=hue1[0];
  hue1[2]=hue1[0];
}


void splitComplimentary()
{

  hue1[0]=abs(hue+106);
  hue1[1]=abs(hue+149);
  hue1[2]=hue1[0];
}

void tertradic()
{

  hue1[0]=abs(hue+63);
  hue1[1]=abs(hue+127);
  hue1[2]=abs(hue+191);
}

void analagous()
{

  hue1[0]=abs(hue+21);
  hue1[1]=abs(hue+42);
  hue1[2]=abs(hue+63);
}




void harmonyMode0()
{
  if(count>1)
  {
    for(int i=baseLed;i<NUM_LEDS;i++)
    {
      int randomHue=hue1[random(0,3)];
      //leds[i] = CHSV(randomHue, 100, 100);
      leds[i].setHue(randomHue);
    }
  }
  else
  {
    for(int i=baseLed;i<NUM_LEDS;i++)
    {
      leds[i] = CHSV(0, 0, 0);
    }
  }
  FastLED.show();
}






void harmonyMode1()
{
  for(int i=0;i<constrain(count,0,NUM_LEDS-baseLed);i++)
  {
    int randomHue=hue1[random(0,3)];
    //leds[baseLed+i] = CHSV(randomHue, 100, 100);
    leds[baseLed+i].setHue(randomHue);
  }
  FastLED.show();
  for(int i=0;i<constrain(count,0,NUM_LEDS-baseLed);i++)
  {
    leds[baseLed+i] = CHSV(0, 0, 0);
  }
  FastLED.show();
}





void harmonyMode2()
{
  for(int i=0;i<constrain(count,0,NUM_LEDS-baseLed);i++)
  {
    int randomHue=hue1[random(0,3)];
    //leds[baseLed+i] = CHSV(randomHue, 100, 100);
    leds[baseLed+i].setHue(randomHue);
  }
  FastLED.show();
  for(int i=0;i<NUM_LEDS-baseLed;i++)
  {
    leds[baseLed+i].fadeToBlackBy( 64 );
  }
  FastLED.show();
}





//GRAPHICS
void mode0()
{
  if(count>1)
  {
    for(int k=baseLed;k<NUM_LEDS;k++)
    {
    // Turn the LED on, then pause
    leds[k].setHue(hue1[0]);
    FastLED.show();
    }
  }
  else
  {
    for(int k=baseLed;k<NUM_LEDS;k++)
    {
    // Turn the LED on, then pause
    leds[k]= CHSV(0, 0, 0);
    FastLED.show();
  }

  }
}





//just on off all the light at once for a pulse
void mode1()
{
   for(int k=0;k<constrain(count,0,NUM_LEDS-baseLed);k++)
  {
  // Turn the LED on, then pause
  leds[baseLed+k].setHue(hue1[0]);
  FastLED.show();
  }
  delay(50);
   for(int k=0;k<constrain(count,0,NUM_LEDS-baseLed);k++)
  {
  // Turn the LED on, then pause
  leds[baseLed+k].setRGB(0,0,0);
  FastLED.show();
  }
}
```

```
//start to light on from previous last lit led
void mode2()
{
   if(count-pre_count > 0)
  {
    for(int k=0;k<=constrain(count-pre_count,0,NUM_LEDS-baseLed);k++)
      {// Turn the LED on, then pause
        leds[baseLed+k].setHue(hue1[0]);
        FastLED.show();
      }
  }
  else
  {
    for(int k=constrain(pre_count,0,NUM_LEDS-baseLed);k>=constrain(count,0,NUM_LEDS-baseLed);k--)
      {
        leds[baseLed+k].setRGB(0,0,0);
        FastLED.show();
      }
  }
}


void mode3()
{
  for(int k=0;k<NUM_LEDS;k+=2)
  {
    leds[k].setHue(hue);
  }

  if(count>20)
  {
    int ran=2*random(0,20)+1;
    for(int k=ran;k<constrain(ran+count,0,NUM_LEDS);k+=2)
    {
      int randomHue=hue1[random(0,3)];
      leds[k].setHue(randomHue);
    }
    FastLED.show();
  }

    for(int k=1;k<NUM_LEDS;k+=2)
    {
    leds[k].fadeToBlackBy( 64 );
    }
    FastLED.show();



}








//COLOUR




//frequency reactive
void mode5()
{

  hue1[0] = (int)(round(abs(x-80)))%255;
  hue1[1]=hue1[0];
  hue1[2]=hue1[0];
}
```

```
//random
void mode6()
{
   hue1[0]   =   random8();
   hue1[1]=random8();
   hue1[2]=random8();
}



//energy reactive
void mode7()
{
hue1[0]= (int)round(map(m,0,20000,0,255));
hue1[1]=hue1[0];
  hue1[2]=hue1[0];
}
```

Main Code With FFT

```
//freq range - 150 Hz to 4300 Hz

#include "arduinoFFT.h"
#include "FastLED.h"
arduinoFFT FFT = arduinoFFT();

#define CHANNEL 4
#define DIGITAL 5

#define NUM_LEDS 60

#define DATA_PIN 2
#define CLOCK_PIN 13

//#define Hue_pot (data pin of knob to change hue)                                  //edit this part when interrupts are available and pots
//#define baseLED_pot (data pin of knob to change no of baseleds)

const uint16_t samples = 128; //This value MUST ALWAYS be a power of 2
const double samplingFrequency = 9000; //Hz, must be less than 10000 due to ADC

unsigned int sampling_period_us;
unsigned long microseconds;

// Define the array of leds
CRGB leds[NUM_LEDS];

/*
These are the input and output vectors
Input vectors receive computed results from FFT
*/
double vReal[samples];
double vImag[samples];

#define SCL_INDEX 0x00
#define SCL_TIME 0x01
#define SCL_FREQUENCY 0x02
#define SCL_PLOT 0x03

double m;
double ave,val=0;
int count=0;
int pre_count = 0;
uint8_t hue;
uint8_t hue1[3];
int baseLed=20;
double Vibe = 1;   //to capture portion of energy of a frequency band.this should be analog Read
```

```
int red,green,blue =0 ;
float x;
int push_button=12;

void setup(){
  Serial.begin(256000);
  pinMode(CHANNEL, INPUT);
  pinMode(DIGITAL,INPUT);
  sampling_period_us = round(1000000*(1.0/samplingFrequency));
 FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
 for(int i=0;i<50000;i++)
 {
   ave = ave+ analogRead(CHANNEL);
 }

 ave = ave/50000;
}

void loop()
{

//taking audio buffer
  microseconds = micros();
  for(int i=0; i<samples; i++)
    {

      vReal[i] = analogRead(CHANNEL)-ave;
      vImag[i] = 0;

      while(micros() - microseconds < sampling_period_us){
        //empty loop
      }
      microseconds += sampling_period_us;
      Serial.println(vReal[i]);
  }

 fourier();

  for(int i=0;i<round(Vibe*sizeof(vReal)/sizeof(double));i++)
   {
      m +=abs(vReal[i]);
   }

  count = round(map(m,0,100000,0,50));

  bassColor();

  selectmode(push_button);

  pre_count=count;
  m=0;

}


int selectmode(int var)
{
  switch (var)
  {
    case 1:
      complimentory();
      harmonyMode0();
      break;

    case 2:
      complimentory();
      harmonyMode1();
      break;

    case 3:
      complimentory();
      harmonyMode2();
      break;

    case 4:
```

```
  splitComplimentary();
  harmonyMode0();
  break;

case 5:
  splitComplimentary();
  harmonyMode1();
  break;

case 6:
  splitComplimentary();
  harmonyMode2();
  break;

case 7:
  tertradic();
  harmonyMode0();
  break;

case 8:
  tertradic();
  harmonyMode1();
  break;

case 9:
  tertradic();
  harmonyMode2();
  break;


case 10:
  analagous();
  harmonyMode0();
  break;

case 11:
  analagous();
  harmonyMode1();
  break;

case 12:
  analagous();
  harmonyMode2();
  break;

case 13:
  mode5();
  mode0();
  break;

case 14:
  mode5();
  mode1();
  break;

case 15:
  mode5();
  mode2();
  break;

case 16:
  mode6();
  mode0();
  break;

case 17:
  mode6();
  mode1();
  break;

case 18:
  mode6();
  mode2();
  break;
```

```
    case 19:
      mode7();
      mode2();
      break;

    case 20:
      mode7();
      mode0();
      break;

    case 21:
      mode7();
      mode1();
      break;

    case 22:
      mode7();
      mode2();
      break;

    case 23:
      complimentory();
      mode3();
      break;

    case 24:
      mode5();
      mode3();
      break;

    case 25:
      mode6();
      mode3();
      break;

    case 26:
      mode7();
      mode3();
      break;

    case 27:
      splitComplimentary();
      mode3();
      break;

    case 28:
      tertradic();
      mode3();
      break;

    case 29:
      analagous();
      mode3();
      break;

  }
}

void fourier()
{
  //fft
  FFT.Windowing(vReal, samples, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
  FFT.Compute(vReal, vImag, samples, FFT_FORWARD); /* Compute FFT */
  FFT.ComplexToMagnitude(vReal, vImag, samples);
 //PrintVector(vReal, (samples >> 1), SCL_FREQUENCY);
  x = FFT.MajorPeak(vReal, samples, samplingFrequency);
}


void bassColor()
{
//  hue=analogRead(Hue_pot);
//  baseLed = analogRead(baseLED_pot);
  hue = 125;
  for(int i=0;i<baseLed;i++)
  {
    //leds[i] = CHSV(hue, 100, 100);
    leds[i].setHue(hue);
  }
}

void complimentory()
{

  if(hue<180)
  {
    hue1[0]=hue+127;
  }
  else
  {
    hue1[0]=hue-127;
  }

  hue1[1]=hue1[0];
  hue1[2]=hue1[0];
}

void splitComplimentary()
{

  hue1[0]=abs(hue+106);
  hue1[1]=abs(hue+149);
  hue1[2]=hue1[0];
}

void tertradic()
{

  hue1[0]=abs(hue+63);
  hue1[1]=abs(hue+127);
  hue1[2]=abs(hue+191);
}

void analagous()
{

  hue1[0]=abs(hue+21);
  hue1[1]=abs(hue+42);
  hue1[2]=abs(hue+63);
}

void harmonyMode0()
{
  if(count>1)
  {
    for(int i=baseLed;i<NUM_LEDS;i++)
    {
      int randomHue=hue1[random(0,3)];
      //leds[i] = CHSV(randomHue, 100, 100);
      leds[i].setHue(randomHue);
    }
  }
  else
  {
    for(int i=baseLed;i<NUM_LEDS;i++)
    {
      leds[i] = CHSV(0, 0, 0);
    }
  }
  FastLED.show();
}

void harmonyMode1()
{
  for(int i=0;i<constrain(count,0,NUM_LEDS-baseLed);i++)
  {
    int randomHue=hue1[random(0,3)];
    //leds[baseLed+i] = CHSV(randomHue, 100, 100);
    leds[baseLed+i].setHue(randomHue);
  }
  FastLED.show();
  for(int i=0;i<constrain(count,0,NUM_LEDS-baseLed);i++)
  {
    leds[baseLed+i] = CHSV(0, 0, 0);
  }
  FastLED.show();
}

void harmonyMode2()
{
  for(int i=0;i<constrain(count,0,NUM_LEDS-baseLed);i++)
  {
    int randomHue=hue1[random(0,3)];
    //leds[baseLed+i] = CHSV(randomHue, 100, 100);
    leds[baseLed+i].setHue(randomHue);
 }
  FastLED.show();
  for(int i=0;i<NUM_LEDS-baseLed;i++)
  {
    leds[baseLed+i].fadeToBlackBy( 64 );
  }
  FastLED.show();
}
//GRAPHICS
void mode0()
{
  if(count>1)
  {
    for(int k=baseLed;k<NUM_LEDS;k++)
    {
    // Turn the LED on, then pause
    leds[k].setHue(hue1[0]);
    FastLED.show();
    }
  }
  else
  {
    for(int k=baseLed;k<NUM_LEDS;k++)
    {
    // Turn the LED on, then pause
    leds[k]= CHSV(0, 0, 0);
    FastLED.show();
  }

  }
}
//just on off all the light at once for a pulse
void mode1()
{
   for(int k=0;k<constrain(count,0,NUM_LEDS-baseLed);k++)
  {
  // Turn the LED on, then pause
  leds[baseLed+k].setHue(hue1[0]);
  FastLED.show();
  }
  delay(50);
   for(int k=0;k<constrain(count,0,NUM_LEDS-baseLed);k++)
  {
  // Turn the LED on, then pause
  leds[baseLed+k].setRGB(0,0,0);
  FastLED.show();
  }
}
//start to light on from previous last lit led
void mode2()
{
   if(count-pre_count > 0)
  {
    for(int k=0;k<=constrain(count-pre_count,0,NUM_LEDS-baseLed);k++)
       {// Turn the LED on, then pause
        leds[baseLed+k].setHue(hue1[0]);
        FastLED.show();
      }
  }
  else
  {
    for(int k=constrain(pre_count,0,NUM_LEDS-baseLed);k>=constrain(count,0,NUM_LEDS-baseLed);k--)
      {
        leds[baseLed+k].setRGB(0,0,0);
        FastLED.show();
      }
  }
}


void mode3()
{
  for(int k=0;k<NUM_LEDS;k+=2)
  {
    leds[k].setHue(hue);
  }

  if(count>20)
  {
    int ran=2*random(0,20)+1;
    for(int k=ran;k<constrain(ran+count,0,NUM_LEDS);k+=2)
    {
      int randomHue=hue1[random(0,3)];
      leds[k].setHue(randomHue);
    }
    FastLED.show();
  }

    for(int k=1;k<NUM_LEDS;k+=2)
    {
    leds[k].fadeToBlackBy( 64 );
    }
    FastLED.show();

}

//COLOUR
//frequency reactive
void mode5()
{

  hue1[0] = (int)(round(abs(x-80)))%255;
  hue1[1]=hue1[0];
  hue1[2]=hue1[0];
}

//random
void mode6()
{
    hue1[0]   =   random8();
    hue1[1]=random8();
    hue1[2]=random8();
}




//energy reactive
void mode7()
{
hue1[0]= (int)round(map(m,0,20000,0,255));
hue1[1]=hue1[0];
  hue1[2]=hue1[0];
}

//void PrintVector(double *vData, uint16_t bufferSize, uint8_t scaleType)
//{
//  for (uint16_t i = 0; i < bufferSize; i++)
//  {
//    double abscissa;
//    /* Print abscissa value */
//    switch (scaleType)
//    {
//      case SCL_INDEX:
//        abscissa = (i * 1.0);
//  break;
//      case SCL_TIME:
//        abscissa = ((i * 1.0) / samplingFrequency);
//  break;
//      case SCL_FREQUENCY:
//        abscissa = ((i * 1.0 * samplingFrequency) / samples);
//  break;
//    }
//    Serial.print(abscissa, 6);
//    if(scaleType==SCL_FREQUENCY)
//      Serial.print("Hz");
//    Serial.print(" ");
//    Serial.println(vData[i], 4);
//  }
//  Serial.println();
//}





//void mode3()
//{
//  float w=0.2168;
//  if(100<x&&x<3000)
//  {
//  red = sin8(w*x);
//  green = sin8(w*x-23.421);
//  blue = sin8(w*x-37.2616);
//  }
//  else{red=100;green=0;blue=0;}
//}


//void mode4()
//{
//  double val = pow(2,40)*x/pow(10,12);
//  double p_red = 100000*exp(-0.5* pow((x - 200)/400 , 2) )/400;
//  double p_green = 100000*exp(-0.5* pow((x - 700)/400 , 2) )/400;
//  double p_blue = 100000*exp(-0.5* pow((x - 1200)/400 , 2) )/400;
//  red = round(p_red);
//  green = round(p_green);
//  blue = round(p_blue);
//  Serial.print(p_red);
//  Serial.print("    ");
//  Serial.print(p_green);
//  Serial.print("    ");
//  Serial.println(p_blue);
//  //Serial.println(val);
//}

//void mode8()
//{
//  fourier();
//  if(x<300)
//  {
//    red = round(map(x,0,300,0,255));
//    green = 0;
//    blue=255-red;
//  }
//
//  else if(x>300 && x<500)
//  {
//
//    green = round(map(x,300,500,0,255));
//      red = 255 -green;
//    blue=0;
//  }
//  else if(x>500 && x<2000)
//  {
//    red = 0;
//    blue= round(map(x,500,2000,0,255));
//    green = 255-blue;
//
```

```
//  }
//  else if(x>2000 && x<3000)
//  {
//    red =  map(x,800,3000,0,255);
//    green =0;
//    blue=255-red;
//  }
//  else
//  {
//    red=0;blue=0;green=0;
//  }
//}

//void mode3()
//{
//
//
//  if(count>1)
//  {
//    for(int k=baseLed;k<NUM_LEDS;k++)
//    {
//    // Turn the LED on, then pause
//    leds[k].setHue(hue1[0]);
//    FastLED.show();
//    }
//  }
//  else
//  {
//    for(int k=baseLed;k<NUM_LEDS;k++)
//    {
//    // Turn the LED on, then pause
//    leds[k].setHue(hue);
//    FastLED.show();
//    }
//  }
//}
```

Plots of Spectrum Areas

```
load('data');

%freq_parts = 2 spectrum is divided in to these two parts
buffer_size=256;
buffer=zeros(buffer_size);
SpectrumAreaOfData=[];
for k=1:size(Data,2)

    SpectrumAreaOfSong =[];
    for j=1:((size(Data,1)-1)/buffer_size)
    x = Data( (j-1)*buffer_size+1:(j-1)*buffer_size+buffer_size ,k);
    d=abs(fft(x));
%     d=d(1:100);
    spectrum_area = [ sum( d(1:size(d,1)/4) )/sum(d)  sum( d( size(d,1)/4 +1 : size(d,1)*2/4 ))/sum(d)  sum( d( size(d,1)*2/4 +1 : size(d,1)*3/4 ))/sum(d)  sum( d(
size(d,1)*3/4 +1 : size(d,1) ))/sum(d)];
    SpectrumAreaOfSong = [SpectrumAreaOfSong ; spectrum_area];
    end
    SpectrumAreaOfData = [SpectrumAreaOfData SpectrumAreaOfSong];
end

plot3(SpectrumAreaOfData(:,2),SpectrumAreaOfData(:,3),SpectrumAreaOfData(:,4),'bo');hold on;
% plot3(SpectrumAreaOfData(:,6),SpectrumAreaOfData(:,7),SpectrumAreaOfData(:,8),'bo');hold on;
plot3(SpectrumAreaOfData(:,18),SpectrumAreaOfData(:,19),SpectrumAreaOfData(:,20),'ro');hold on;
plot3(SpectrumAreaOfData(:,34),SpectrumAreaOfData(:,35),SpectrumAreaOfData(:,36),'go')
legend('pop','rap','rock');
grid on;
```

FFT Algorithm

```
#include <string>
#include <sstream>
```

```
#include <iostream>

#include <fstream>
#include<math.h>
using namespace std;
int const NoOfSample=128;
void fft(float X[][2],float fourier[][2]);
void odd(float y[][2],float oddout[][2]);
void even(float z[][2],float evenout[][2]);
float maximum(float arr[][2],float * freqval);

main()
{
      int sample_count=0;
      float buff_mic[NoOfSample][2];
     //readcsv and check code
     ifstream myfile;
      myfile.open("file.csv");

      for(int h=0;h<NoOfSample;h++)
      {
        string line;
        getline(myfile,line);
        buff_mic[h][0]=std::stof(line);
        buff_mic[h][1]=0;
      }

      float FourierOutput[NoOfSample/2][2];

      fft(buff_mic,FourierOutput);

//finding the max val
      float AmplitudeofFreq;
      float freqIndex = maximum(FourierOutput,&AmplitudeofFreq);
      cout << AmplitudeofFreq <<endl;
}
void odd(float y[][2],float oddout[][2])
{

  int rows=sizeof(y)/sizeof(y[0]);      //int cols=sizeof(y[0])/sizeof(y[0][0]);      //float oddarray[rows][cols];

    for(int i=1;i<rows;i=i+2)
    {
      oddout[i][0]=y[i][0];
      oddout[i][1]=y[i][1];
    }
    //return oddarray;
}
void even(float z[][2],float evenout[][2])
{
  int rows=sizeof(z)/sizeof(z[0]);        //int cols=sizeof(z[0])/sizeof(z[0][0]);      //float evenarray[rows][cols];

    for(int i=0;i<rows;i=i+2)
    {
      evenout[i][0]=z[i][0];
      evenout[i][1]=z[i][1];
    }
    //return evenarray;
}
void fft(float X[][2],float fourier[][2])
{

  int rows=sizeof(X)/sizeof(X[0]);
  //int cols=sizeof(X[0])/sizeof(X[0][0]);

  if(rows==1)
  {
    fourier[0][0]= X[0][0];
    fourier[0][1]= X[0][1];
  }
  else
  {
    float oddcomponents[rows/2][2];
    float evencomponents[rows/2][2];
    odd(X,oddcomponents);
    even(X,evencomponents);

    float oddFourier[rows/2][2];
    fft(oddcomponents,oddFourier);

    float evenFourier[rows/2][2];
    fft(evencomponents,evenFourier);


    //float fftout[rows][cols];

    for(int k=0;k<rows/2;k++)
    {
      float angle=2*3.14*k/rows;
      fourier[k][0] = evenFourier[k][0]+cos(angle)*oddFourier[k][0]-sin(angle)*oddFourier[k][1];
      fourier[k][1] = evenFourier[k][1]+cos(angle)*oddFourier[k][1]+sin(angle)*oddFourier[k][0];

      fourier[rows/2+k][0] = evenFourier[k][0]-cos(angle)*oddFourier[k][0]+sin(angle)*oddFourier[k][1];
      fourier[rows/2+k][1] = evenFourier[k][1]-cos(angle)*oddFourier[k][1]-sin(angle)*oddFourier[k][0];

    }
    //return fourier;
  }
}


float maximum(float arr[][2],float * freqval)
{
  int index;
  freqval=0;
  int rows=sizeof(arr)/sizeof(arr[0]);
    for(int u=0;u<rows;u++)
    {
      if(pow(*freqval,2)<(pow(arr[u][0],2)+pow(arr[u][1],2)))
      {
        float freqval=sqrt(pow(arr[u][0],2)+pow(arr[u][1],2));
        index = u;
      }

    }
    return index;
}
```

DFT Algorithm

```
#include<stdio.h>
#include<stdint.h>
#include <complex.h>
#define _USE_MATH_DEFINES // for C
#include <math.h>

void plot(double x,double f);
void dft(double z[],double complex ft[],int N);
int main()
{
    FILE *data = NULL;
    FILE *gnupl = NULL;
    int n = 44100;
    int buff_size=n;
    data=fopen("data.tmp","w");                    //for gnuplot
    gnupl=_popen("gnuplot -persitent","w");
    double sampling_freq=44100;
    double Ts=1/sampling_freq;
    //time
    double t[n];
    for(int i=0;i<n;i++)
    {
        t[i]=i*Ts;
        //printf(" %lf \n",t[i]);
    }
    //signal getting from ffmpeg throygh piping
    double y[buff_size];
    int16_t buff[buff_size];
    FILE *pipein;
    pipein = _popen("ffmpeg -i A5.wav -f s16le -ac 1 -", "r");
    fread(buff, 2 ,buff_size, pipein);
    _pclose(pipein);
//converting to double to send to dft
   for(int j=0;j<buff_size;j++)
   {
        //y[j]=20*sin(2*M_PI*10*j/n)+10*sin(2*M_PI*1*j/n);
        y[j]=(double)buff[j];
       //printf(" %lf %d\n",y[j],buff[j]);
   }
//-----------------------------------------------------------------------------------------------------------------------------------------
//Fourier
    double complex ft[n];              //minimum freq corressponds with buffer size.to denote minimum frequency we need 441000/(1/n) samples in freq domain
    dft(y,ft,n);
//absolute vale for plotting purposes
    for(int k=0;k<n/2;k++)
    {
        ft[k]=cabs(ft[k]);
       //printf(" %lf +i %lf \n ",creal(ft[k]),cimag(ft[k]));
    }
//saving data to a file "data" to be send to gnuplot
   for(int u=0;u<4000;u++)
   {
       //double fr_steps=u/(n/sampling_freq);
       fprintf(data,"%d %lf \n",u,ft[u]);
       //plot(ft[u],u);
   }
double fun_freq=1/(n/sampling_freq);
printf("FUndamental Frequency - %lf",fun_freq);
//gnuplot
    fprintf(gnupl,"%s \n","set title\"demo\"");
    fprintf(gnupl,"%s \n","plot 'data.tmp' with boxes");

    return 0;
}
void dft(double *z,double complex * ft,int N)
{
    double complex q=(-2*M_PI/N);

    for(int i=0;i<4000;i++)
    {
        double complex s=0;
        for(int j=0;j<N-1;j++)
        {
            s=s+(cos(q*i*j)+sin(q*i*j)*I)*z[j];
        }
        //printf("%f + %f i\n",crealf(s/N),cimagf(s/N));
        ft[i]=s/N;
    }
}
void plot(double x,double f)
{
    printf("%lf   ",f);
    double i=0;
    while(i<x)
    {
        printf("*");
        i=i+.5;
    }
    printf("\n");

}
```

Matlab simulation code

```
% Load wave file
[x, Fs] = audioread('piano-f_F_major.wav');


% Parameters
x = x(:,1);
dimen = size(x);
q0 = zeros(1, 39);
q1 = zeros(1, 39);
q2 = zeros(1, 39);

% Frequencies of notes
FA2 = 110.00;
FAs2 = 116.54;
FB2 = 123.47;

FAb3 = 207.65;
FA3 = 220.00;
FBb3 = 233.08;
FB3 = 246.94;
FC3 = 130.81;
FCs3 = 138.59;
FD3 = 146.83;
FEb3 = 155.56;
FE3 = 164.81;
FF3 = 174.61;
FFs3 = 185.00;
FG3 = 196.00;

FAb4 = 415.30;
FA4 = 440.00;
FBb4 = 466.16;
FB4 = 493.88;
FC4 = 261.63;
FCs4 = 277.18;
FD4 = 293.66;
FEb4 = 311.13;
FE4 = 329.63;
FF4 = 349.23;
FFs4 = 369.99;
FG4 = 392.00;

FAb5 = 830.61;
FA5 = 880;
FBb5 = 932.33;
FB5 = 987.77;
FC5 = 523.25;
FCs5 = 554.37;
FD5 = 587.33;
FEb5 = 622.25;
FE5 = 659.25;
FF5 = 698.46;
FFs5 = 739.99;
FG5 = 783.99;

% Note names
names = ["A2", "A#2", "B2", "A#3", "A3", "Bb3", "B3", "C3", "C#3", "D3", "Eb3", "E3", "F3", "F#3", "G3", "A#4", "A4", "Bb5", "B4", "C4", "C#4", "D4",
"Eb4", "E4", "F4", "F#4", "G4", "A#5", "A5", "Bb5", "B5", "C5", "C#5", "D5", "Eb5", "E5", "F5", "F#5", "G5"];

% Average power
prob = zeros(size(names));

% Normalizing
w = [FA2, FAs2, FB2, FAb3, FA3, FBb3, FB3, FC3, FCs3, FD3, FEb3, FE3, FF3, FFs3, FG3, FAb4, FA4, FBb4, FB4, FC4, FCs4, FD4, FEb4, FE4, FF4, FFs4,
FG4, FAb5, FA5, FBb5, FB5, FC5, FCs5, FD5, FEb5, FE5, FF5, FFs5, FG5];
w = (2 * pi) .* (w ./ Fs);
for n=2:dimen
  for j=1:39
```

```
        q0(j) = x(n) + 2 * cos(w(j)) * q1(j) - q2(j);
        q2(j) = q1(j);
        q1(j) = q0(j);
    end
    psdMag = sqrt((q1 .* q1) + (q2 .* q2) - (2 .* cos(w) .* q1 .* q2));
    prob = 0.7 * prob + 0.3 * (psdMag / max(psdMag));
    [M, I] = max(prob);
    if mod(n, 1000) == 0
     disp(names(I));
     prob = zeros(size(names));
    end
end
```