Music Detecting Light System

Team 72 - Francis Mui, Ran Wang, and Alfredo Sanchez

ECE 445 Design Document- Spring 2020

TA: Ruhao Xia

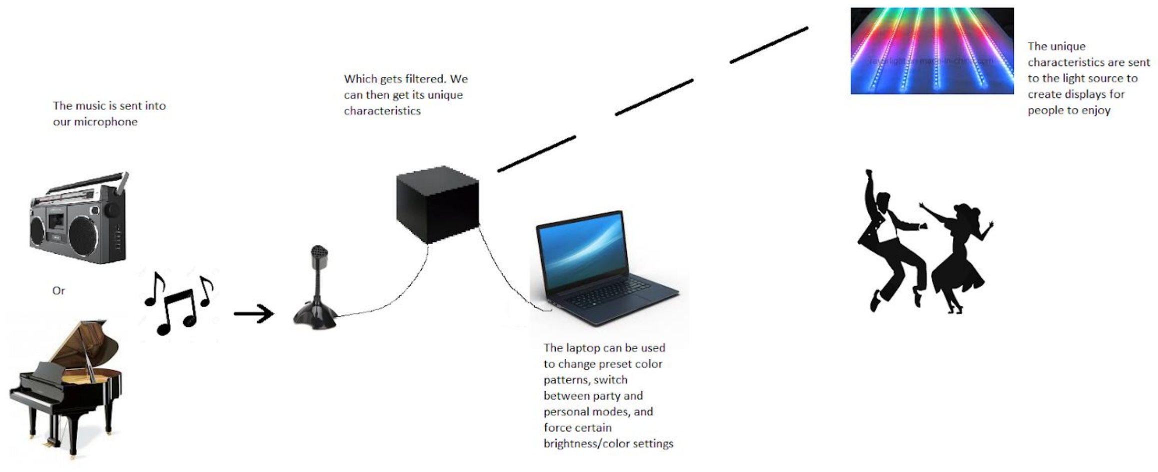
1. INTRODUCTION

1.1 Problem and Solution Overview.

The system objective is for the light patterns to change based on the music detected and on the user's chosen settings. For example, when used in a party and a strong music is detected, the lights are supposed to react to the "mood" of the music, showing strong synchronized beats, with fast changing, enthusiastic color patterns. When in personal use, such as when the user is playing jazz music, the lights change to a colder, background color to reflect the genre of the music, creating a more immersive atmosphere. So if the rhythm, genre and change of pace of the music is accurately reflected and the user's preferences are met, the system is considered successful.

Lights and music are two features that are very powerful in providing entertainment. For example, party hosts often provide rock music and warm-color lighting to improve the atmosphere. Also, musicians would sometimes relate a piece of classical or jazz music to a specific color. The problem is the colors and the music itself are often separated. A system providing synchronous lighting to the music would be an interesting solution for individuals who frequently listen to music and want a matching atmospheric lighting. This will also extend to people who host home parties and are looking for a way to make it more exciting or for people who desire to feel more immersed when listening to any kind of music. The key point is that people will be able to create a synchronous ambience combining both music and lights.

1.2 Visual Aid


The music is sent into our microphone
Or
Which gets filtered. We can then get its unique characteristics
The laptop can be used to change preset color patterns, switch between party and personal modes, and force certain brightness/color settings
The unique characteristics are sent to the light source to create displays for people to enjoy

1.3 High-level requirements list

1.3.1. Detecting and recognizing the music pitch.

The high level goal for this system is for the light patterns to change based on the music detected. In order to do that the system must detect the type of music that is being played and classify it in order to form the proper output, using parameters such as genre, chord progression, frequency, and strong beats.

1.3.2. Controlling lights.

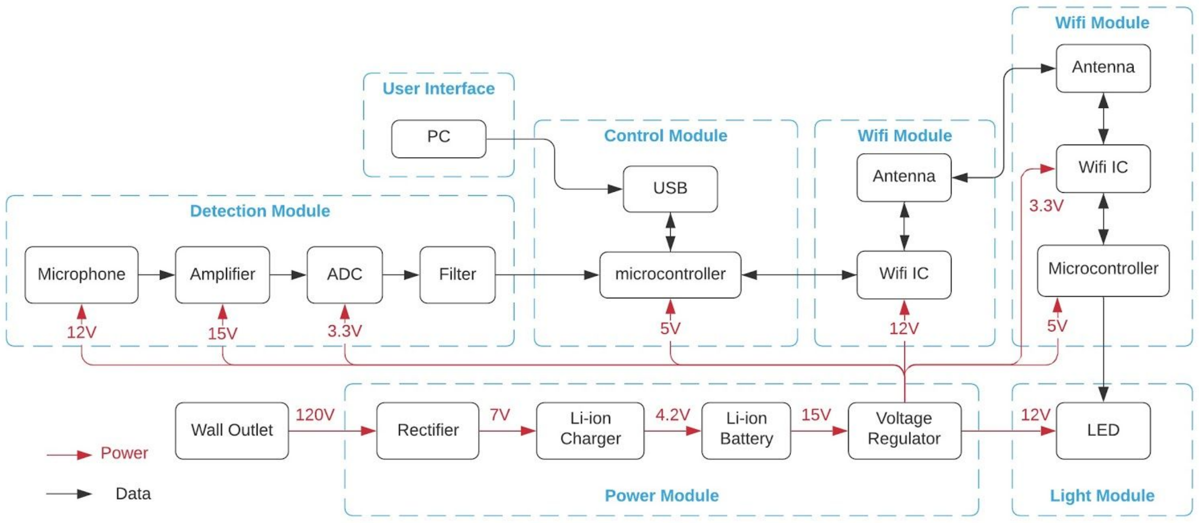
The other part of the main goal of the project is the lights being able to change. The only way of doing that is by controlling the pattern of the lights, so the system must have total control of the light outputs.

1.3.3. User choosing preset patterns.

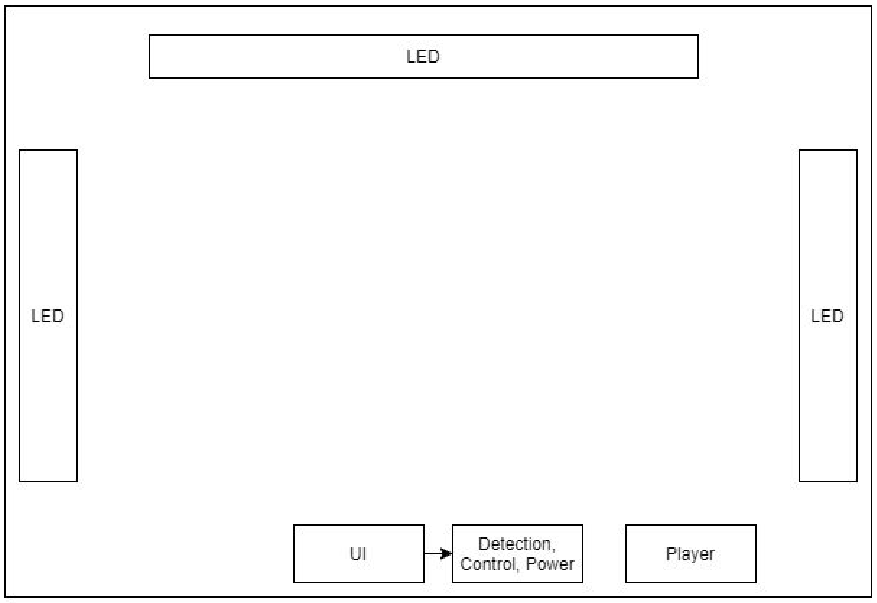
The project must also have some user interface software so the user can choose to use either a preset light pattern or one that is custom set by the user even without having any music playing.

2. DESIGN

2.1 Block Diagram



2.2 Physical Design



Top view of the system setup (generalized)

The microphone is installed near the music player to reduce the noise received. LED light bands are installed around the room and on the ceiling to create light patterns. The control, Wifi and user interface modules can be installed at any convenient places at will, and does not interfere much with the users when in use. (for example, in a crowded party)

2.3. Detection Module

The Detection Module will need to be able to read the audio signal, amplify it and convert it from analog to digital, preliminarily filter out the background noises, and send this new amplified signal to the Control Module. This needs to be active while the machine is on at all times, and needs to quickly identify when the song has changed. The microphone itself will need 12V to activate. This module contains a microphone to record the analog audio signal from the room where it's installed, an amplifier to increase its strength, a digitizer to perform an analog-to-digital conversion, and a digital filter to preliminarily eliminate the background noises recorded. Recognition of the music is performed in the recognition module followed.

Requirement	Verification
Amplifier 1. Accepts audio signal from microphone, and amplifies it for processing.	(a) Receive data from the microphone with around 50mV voltage. (b) Use oscilloscope to verify the amplified voltage level is around 2.0-3.4 V.
Filter 1.Filter out background noises outside of the desired 50-5k Hz frequency range.	(a) Record audio samples. (b) Perform a DTFT to see the filter outcome. (c) The noise spectrum should be attenuated to below -20 dB.
Microphone 1. Steady continuous performance at working conditions of 5V and 500mA. 2. Record sound signals of 30-75dB clearly.	1. (a) Connect microphone to the required voltage and current. (b) Record audio signal for at least 3 hours, and verify the audio's continuously recorded. 2. (a) Record audio signal with the desired intensity. (b) Compare the spectra, verify the recorded data's conformity with the original signal.
Analog-Digital Converter 1. Process signals at 2.0-3.4V level to 10-bit digital output.	(a) Send a signal with varying amplitude in the 0-3.4V range and varying frequency of 50-5k Hz.

	(b) Verify a conforming digital output to the original signal's shape.

2.4 Control Module

The Control Module needs to be able to read the signal acquired from the Detection Module and convert it into a signal which accurately conveys the genre, chord progression, frequency and strong beats. It then needs to use these parameters to pick out the appropriate light patterns, either from the preset list or from one obtained from the User Interface, and send it to the Wifi Module. We intend to use either a microcontroller of, if we need a more powerful chip, a raspberry pi. Much like the Detection Module, this needs to be active at all times to identify the song, but will only need to send outputs to the Wifi Module when the light is turned on. Both of which require the Power Module to provide approximately 5V of power, +/- 0.5V.

This module is the part that is in charge of recognizing the music, receiving user commands from the user interface, and controlling the light module accordingly. After the microprocessor receives the audio signal, it performs the adaptive noise cancellation, audio fingerprinting algorithm and pitch detection to recognize the music's genre, frequency and strong beats. After the recognition, it loads the matching preset light pattern, along with potential user commands to change preferred patterns (explained later), and sends control signal to the Wifi module.

Requirement	Verification
Raspberry Pi Microcontroller 1. Communicate with the user interface app with less than 500ms latency.	(a) Connect PC to microcontroller through USB. (b) Send commands through the PC app, receive feedback. (c) Verify the latency is within the acceptable level.

2.5 Wifi Module

The Wifi Module will need to receive the signal from the Control Module and immediately send the signal to the light Module. This will only activate when we need the lights to turn on, and otherwise will block inputs. This needs to be powered by the Power Module with anywhere between 1.8 and 5 V. This module is added to enable remote connection between the control module and light module. Since wiring the lights directly with the control module can pose a lot of problems when physically setting up the system, remote controlling will allow more flexibility. We will also need to add another small controlling system, in which we will use a microchip in order to obtain the array of bits calculated or obtain from the raspberry pi. It would be a microcontroller embedded into the light module that will receive the data from the wifi module and send it to the light as an output.

Requirement	Verification
Antenna 1. Matched impedance of 50 Ohms +/- 5% at 2.4GHz working frequency.	(a) Test the antenna impedance with an antenna impedance meter. (b) Verify it's within the IEEE standard.
Wifi IC 1. Communicate over UART protocol.	(a) Connect to a Wifi chip's UART port and a computer. (b) Program the IC to send control the LED behavior. (c) Observe and verify the correct behavior of the LED.
Microcontroller 1. Embedded at the LED side. Receive data and control lights with less than 200ms latency.	(a) Send command from control module. (b) Verify the Wifi module's reception. (c) Lights perform tasks with acceptable latency.

2.6 Light Module

The Light Module will need to receive the signal from the Wifi Module and output the signal as varying lights. They will only turn on when they receive inputs from the Wifi Module This needs to be powered by the Power Module with 12 V at 0.6 A per meter of LEDs. Multicolor LED light bands to be set up around the room and perform the desired task: a lighting that matches the music being played. Receives digital control signals through the Wifi module and changes the colors accordingly.

Requirement	Verification
Multicolor LED light band 1. Steady output of multicolor light with 12V, 600mA input power for at least 3 hours. 2. Temperature should be below 125 degrees Celsius.	1. (a) Connect to 12V, 600mA power and control signals. (b) Observe and verify steady brightness and color changing for at least three hours. 2. (a) During operation, verify with a thermometer that the temperature of the light is below 125 degrees Celsius.

2.7 User Interface

The User Interface needs to have sliders/buttons that determine how active a person would want their lights to be, from whether they want the colors to change dramatically for party settings to changing gradually for personal settings or for concerts. This would also have options to place emphasis on certain colors or brightness. This would come with presets for party and personal use, and this signal would be outputted to the Control Module to determine what color patterns would be most appropriate. The personal computer is connected to the control module through a USB port. There will be an implementation of a laptop app for the user to choose their preferred light colors and patterns, to switch between personal mode (for music appreciation, etc.; gradual change, colder colors) and party mode (for enthusiastic music played in a party; emphasizes strong beat effects), and to customize other settings such as brightness and favorite color is necessary.

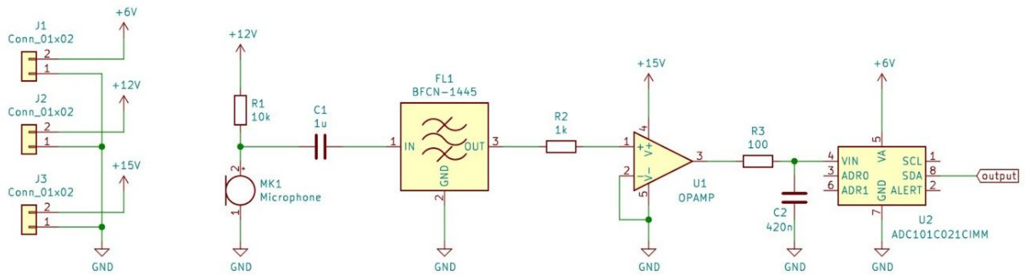
2.8 Power Module

The Power Module needs to be able to power the Detection, Control, Wifi, and Light Module. This will need to provide power for at least 3 hours so that the system can stay on during an entire concert or the majority of a party. It contains a rechargeable battery that draws power from the wall outlet, and a power distribution unit to power the other modules from the battery. Since the system is designed to allow for convenient movement and installment for the user when looking for entertainment, a central power module would be preferable than plugging modules in the outlet separately.

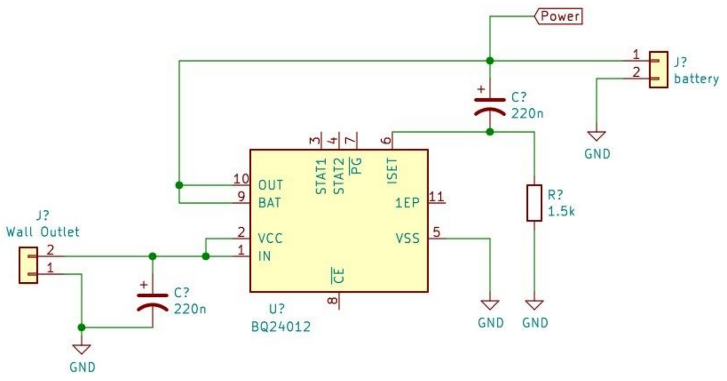
Requirement	Verification
Linear Rectifier 1. Convert wall outlet AC voltage to a steady DC voltage.	(a) Connect rectifier to a load. (b) Power the circuit from the wall outlet. (c) Use oscilloscope to verify the output of the rectifier is 120V DC within 3% tolerance.
Dropout positive voltage regulator. **Low Quiescent Current LDO** 1. Vout = 3.3 V ± 0.3 V at 100 μA	**11.1 V DC to 3.3 V DC** **Switch-Mode Regulators** 1. Verification Process for Item 1: (a) Attach 33 k⌦ resistor as load (b) Attach oscilloscope across load (c) Supply regulator with 11.1 V DC (d) Ensure output voltage remains 3 V and 3.6 V
Li-ion Battery 1. Provides steady output of 15V, 600mA for at least 3 hours without charging.	(a) Fully charge the battery. (b) Disconnect from charger. (c) Connect to circuit. (d) Test to verify a stable 15 V, 600mA output within 5% tolerance for 3 hours.
Li-ion Charger	1. (a) Discharge battery.

1. Steady charging output of 4.2V +/- 5% and 500mA +/- 3%. 2. Stable working temperature below 125 degrees Celsius.	(b) Connect to charger with an input of 7V. (c) Verify the charging voltage, current levels are within the designated levels. 2. (a) During charging, use a thermometer to measure and verify the charger’s temperature.

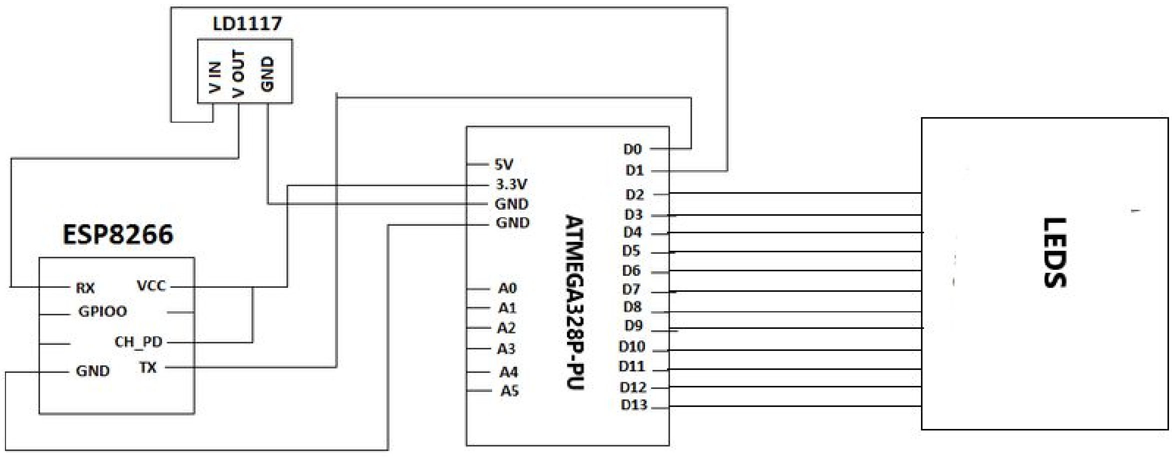
2.9 Schematics



Detection Module Schematic



Power Module Schematic



Simple Wifi Module Schematic.

2.10 Software

- **Changing from Analog to Digital.**

As everyone knows sound is a vibration that propagates as a waveform based on pressure and that moves itself through a medium such as air or water. When the vibration reaches out ears, it moves three little bones (known as malleus, incus and stapes) that transmits the vibration to some other cells located in the inner ear. Furthermore, those cells produce electrical impulses that are transmitted to our brain through the auditory nerve of the ear.

The procedure is commonly imitated by most recording devices, they convert sound as a waveform into an electronic signal. In a microphone, the first electric component that is found is transferred as a continuous analog signal, which in the digital world will be useless, so it has to be changed into a discrete signal which is easier to be processed digitally. This procedure will be done by capturing digital values that will represent the signal amplitude.

Conversion implicates quantization of the entrance quantification that will introduce some error. As matter of fact, instead of a simple conversion we will use an analog-to-digital converter that is a device that converts an analog signal into a digital signal. The analog-to-digital converter divides the signal in tiny pieces and converts those pieces using a process known as sampling.

Nyquist-Shannon Theorem establishes a sufficient condition for a sample rate that permits a discrete sequence of samples to capture all the information from a continuous-time signal of finite bandwidth. In particular, if we want to capture all the frequencies the human being ear can hear, we must sample the signal at a frequency doubling the humans hearing range. Humans ear can detect frequencies between 20 Hz to 20000 Hz. As a result, most of the time recorded audio normally has a sample rate of 44100 Hz, that will be frequency sample rate that will be used in the project.

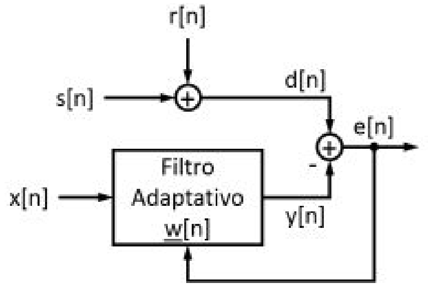
- **Recording and capturing sound.**

Recording an audio signal should not be very difficult, we will use a Raspberry Pi, because it will be powerful enough to handle a microphone, capture the sound and implement the whole algorithm afterwards. We will use a USB microphone that is compatible with the Raspberry Pi we will be using. Here is an example of the code using python of this part:

```
import pyaudio

import wave

form_1 = pyaudio.paInt16 # 16-bit resolution

chans = 1 # 1 channel

samp_rate = 44100 # 44.1kHz sampling rate

chunk = 4096 # 2^12 samples for buffer

record_secs = 30 # seconds to record

dev_index = 2 # where the device is (in index) (use
p.get_device_info_by_index

wav_output_filename = 'test1.wav' # name of .wav file



audio = pyaudio.PyAudio() # create pyaudio instantiation



# create pyaudio stream

stream = audio.open(format = form_1,rate = samp_rate,channels =
chans, \

                    input_device_index = dev_index,input = True, \
                    frames_per_buffer=chunk)
print("recording")
frames = []

# loop through stream and append audio chunks to frame array
for ii in range(0,int((samp_rate/chunk)*record_secs)):
    data = stream.read(chunk)
    frames.append(data)

print("finished recording")

# stop the stream, close it, and terminate the pyaudio instantiation
stream.stop_stream()
stream.close()
audio.terminate()

# save the audio frames as .wav file
wavefile = wave.open(wav_output_filename,'wb')
wavefile.setnchannels(chans)
wavefile.setsampwidth(audio.get_sample_size(form_1))
wavefile.setframerate(samp_rate)
wavefile.writeframes(b''.join(frames))
wavefile.close()
```

- **Filtering the signal to isolate from noise.**

Next step will be isolating noise in order to get a clear signal of just the sound of the music, in order to that we will use the Least Mean Squares Filter (LMS) algorithm which is an adaptive filter commonly used to minimize the ambient noise. Adaptive systems are used to identify systems, predict, cancel noise, cancel eco and equalize a sound adaptively. Our objective in this case will be constructing the adaptive system to cancel background noise from the audio recorded. Block diagram of the adaptive filter would be:



Where s[n] is the audio signal without the background noise; r[n] the background noise captured when recording the audio signal s[n]; d[n] = s[n]+r[n] is the recorded audio with the background noise, which we want to cancel; x[n]= $\tilde{r}$[n] is the noise, correlated with r[n], which we need to adaptate the system to estimate r[n]; w[n] are the M+1 coefficients of the adaptive filter of the adaptive filter for the n time; y[n]= = $r\hat{}$[n] is the estimated noise; and e[n] = d[n]-$r\hat{}$[n]= $s\hat{}$[n] is the error of the adaptive system and the estimated signal of audio without background noise.

We can use as a background noise signal a predefined one of people talking or just some kind of noise that might be similar for the background noise of each ambient. We will also use M=10 and μ=0.0121 as parameters for the adaptive filter. In order to implement the algorithm, we will use the following matlab code, which will be easy to convert into python code by using libermate library.

The algorithm will be:

We will get both the d_n signal that will be the recorded sound and the background sound signal, that will be called as x_n:

```
[d_n,fs]=audioread('PDS_P8_LE2_G2_d_n.wav');
[x_n,fs]=audioread('PDS_P8_LE2_G2_x_n.wav');
```

To begin with, we will build the signals that we will introduce all of them initialized to zero with the desired length, we will create a new_x signal that will be equivalent to the updated x signal, which is updated in the first step of each iteration.

```
x_updated =zeros(M+1,1);
```

The next variable that we will create would be the W matrix, which will have M+1 rows and the length of x_n as columns, for that, we will get the length of x_n and initialize the matrix to zeros as we said before.

```
N=length(x_n);
w=zeros(M+1,N);
```

We will also create a variable y_n, it will be calculated in the second step of each iteration, which will be the one that is subtracted from the d_n signal in each iteration to obtain the e_n error signal. Both signals e_n and y_n will be vectors of N zeros.

```
y_n=zeros(N,1);
e_n=zeros(N,1);
```

Once we have created all the variables that we will use, we will develop the algorithm, in order to that, as we have introduced before, we will follow the steps described below:1) x_updated will be updated with the the sample that is received every time. 2) We will calculate the output sample with y_n=w_n(transpouse)*x_n. 3) We will calculate the error signal that as we have said before it will be the subtraction of d_n minus y_n and that signal after the subtraction and after all the iteration, would be the clean music audio. 4) We will update the coefficients of the filter following the approximation of the gradient with the formula: w(n+1)=w(n)+M+1+2*μ*e_n+x_n.

```
for n = 1:1:N
    j=0;
    for i = n:-1:(n-M)
        if i >= 1
            % Step 1.(Updated x)
            x_updated(M+1-j) = x_n(i);
        else
            % % Step 1.(Updated x)
            x_updated(M+1-j) = 0;
        end
            j = j+1;
    end
```

```
        %Step 2.(We will calculate the output signal
y_n)
              y_n(n)= w(:,n)'*x_updated;
        %Step 3. (We will calculate the signal error
e_n)
        e_n(n)= d_n(n)-y_n(n);
        % Step 4. (We will update w)
        w(:,n+1)= w(:,n) + 2*mu*e_n(n)*x_updated;
end
```

We will obtain in the error signal e_n a similar signal to a clean audio, we will not get the perfect clean music audio, but at least we will remove most of the background sound and the noise left will be enough for classifying the music sound recorded.

- **Time and frequency domain.**

What we got from the program will be a wav file, which in other words is a bytes matrix in time domain. The domain signal in time domain represents the change in amplitude over time.

As Jean-Baptiste Joseph Fourier discovered, it is possible to represent any signal in the time domain by simply giving the set of frequencies, amplitude and phases corresponding to each sinusoid that composes the signal. The representation is known as frequency domain and in a way, the frequency domain acts as a type of fingerprint or signature for the time domain signal, providing a static representation of a dynamic signal.

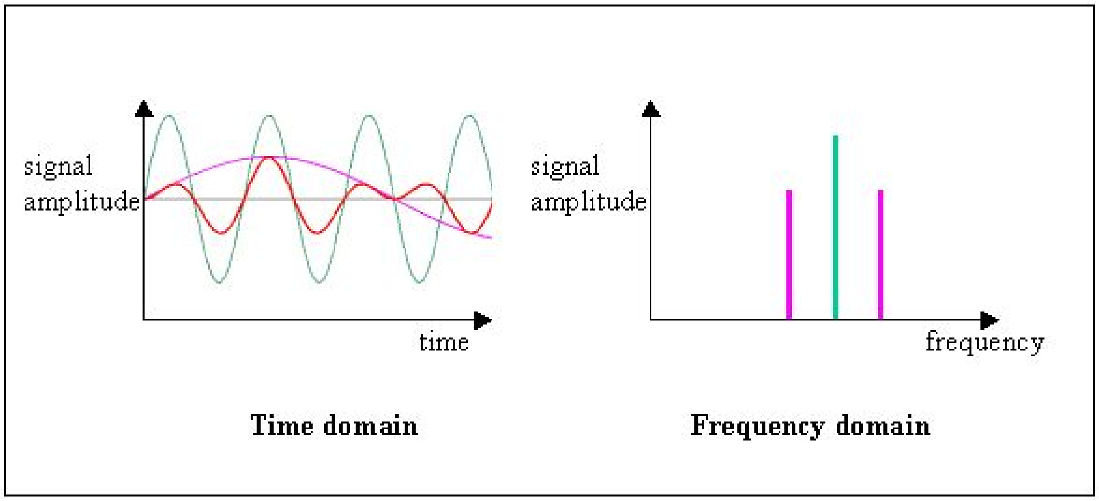


Figure: Time Domain vs Frequency Domain[1]

Analyzing a signal in frequency domain simplifies things immensely. In digital signal processing it is commonly used because engineers can study the spectrum, which is the representation of the signal in the frequency domain, and determine the frequencies that are present and those that are missing. Also it is easier to filter, increase or decrease some frequencies, or recognize the exact tone of the frequencies, which is what we are looking in this project.

- **Discrete Fourier Transform (DFT).**

Therefore, we need to find a way to convert our signal from time domain to frequency domain. The next step would be using the discrete Fourier Transform (DFT). DFT is a mathematical method to perform Fourier Analysis on a discrete sample of the signal. DFT converts a finite list of equidistant samples of a function into the list of the coefficients of a finite combination of complex sinusoids, ordered by their frequencies, considering whether the sinusoids had been sampled in the same proportion.

One of the most popular numeric algorithms for the calculation of DFT is the Fast Fourier transform (FFT). A variation of the FFT Cooley-Tukey algorithm is the most utilized one. The algorithm is based on the divide-and-conquer algorithm, it recursively divides the DFT in multiple and little DFTs. While using the evaluation of a DFT directly requires O(N^2) operations, with the Cooley-Tukey FFT it is computed in O(nlog(n)) operations, making the algorithm faster and simpler. Here is an example of a DFT function in code using python using the NumPy library:

```python
def FFT_vectorized(x):

    """A vectorized, non-recursive version of the
Cooley-Tukey FFT"""

    x = np.asarray( x, dtype=float)

    N = x.shape[0]


    if np.log2(N) % 1 > 0:

        raise ValueError("size of x must be a power of 2")
    # N_min here is equivalent to the stopping condition 
above,

    # and should be a power of 2

    N_min = min(N, 32)


    # Perform an O[N^2] DFT on all length-N_min sub-problems 
at once

    n = np.arange(N_min)

    k = n[:, None]

    M = np.exp(-2j * np.pi * n * k / N_min)

    X = np.dot(M, x.reshape((N_min, -1)))


    # build-up each level of the recursive calculation all 
at once

    while X.shape[0] < N:

        X_even = X[:, :X.shape[1] / 2]

        X_odd = X[:, X.shape[1] / 2:]

        factor = np.exp(-1j * np.pi * np.arange(X.shape[0])

                        / X.shape[0])[:, None]

        X = np.vstack([X_even + factor * X_odd,

                       X_even - factor * X_odd])


    return X.ravel()
```

- **Music Recognition: Fingerprints in a Song.**

A side effect of FFT is that we lose a lot of information about synchronization (although theoretically this can be avoided, the overhead performance is huge) for a 3-minute song, we can see all the frequencies and their magnitudes, but we do not know when those frequencies and magnitudes appeared in the song. We will need to identify when each frequency appears.

That is the reason why we should introduce a sliding window and a fragmentation of the data so then we will transform each piece of information. The size of each fragment could be defined in a lot of ways. For example, if we want to record the stereo sound, with samples of 16 bits, using 44100 Hz, one second of the sound will be of 44100samples*2bytes*2canals ≈ 176 kB. If we choose 4kB for each segment size, we will have 44 pieces of data to analyze for each second of a song, that will be enough for a detailed analysis.

Once we have the information about the frequency of the signal, we can start making the digital footprint of the song and classify them in their genre. In this case we will have to use time and frequency domain information to extract features such as Spectral Rolloff, Spectral Flux, Time Domain Zero Crossing, Mel-Frequency Cepstral Coefficients, Analysis and Texture Window or Rhythmic Content Features.

The features that we will use are those used to represent timbral texture and are based on standard features proposed for music-speech discrimination and speech reocgnition.[2]

The first feature to be calculated will be the Spectral Centroid, which is defined as the center of gravity of the magnitude spectrum of the DFT calculated before. The formula that we will use will be:

$$Ct = \sum_{n=1}^{N} Mt[n] * n / \sum_{n=1}^{N} Mt[n]$$

Where in this case Mt[n] is the magnitude of the DFT at frame t and frequency n. The Spectral Centroid is a measure of spectral shape and higher centroid esteems correspond to “brighter” textures with more high frequencies.

The formula explained before will be implemented in python with the library librosa[3] with the following code:

```
spectral_centroids = librosa.feature.spectral_centroid(x,
sr=sr)[0]

spectral_centroids.shape

(775,)

# Computing the time variable for visualization
```

```
frames = range(len(spectral_centroids))

t = librosa.frames_to_time(frames)

# Normalising the spectral centroid for visualisation

def normalize(x, axis=0):

    return sklearn.preprocessing.minmax_scale(x, axis=axis)

#Plotting the Spectral Centroid along the waveform

librosa.display.waveplot(x, sr=sr, alpha=0.4)

plt.plot(t, normalize(spectral_centroids), color='r')
```

We will also calculate Spectral Rolloff, which is defined as the frequency Rt below which 85% of the magnitude distribution is concentrated. It is another feature that measures spectral shape and the formula that we will use will be:

$$\sum_{n=1}^{Rt} Mt[n] = 0.85 * \sum_{n=1}^{N} Mt[n].$$

That in python code using the same librosa library will be:

```
spectral_rolloff = librosa.feature.spectral_rolloff(x+0.01,
sr=sr)[0]

librosa.display.waveplot(x, sr=sr, alpha=0.4)

plt.plot(t, normalize(spectral_rolloff), color='r')
```

We will calculate the Mel-Frequency Cepstral Coefficients that are defined as the small set of features which concisely describe the overall shape of a spectral envelope, in other words, it models the characteristic of the human voice. The procedure basically consists of after taking the log-amplitude of the magnitude spectrum, the FFT bins are grouped and smoothed according to the perceptually motivated Mel-frequency scaling. Finally, in order to decorrelate the resulting feature vectors a discrete cosine transform is performed. Although typically 13 coefficients are used for speech representation, we have found that the first five coefficients provide the best genre classification performance. A piece of code calculating the MFCCs of an audio signal:

```
mfccs = librosa.feature.mfcc(x, sr=fs)
```

We can also perform feature scaling such that each coefficient dimension has zero mean and unit variance that will be more accurate:

```
import sklearn

mfccs = sklearn.preprocessing.scale(mfccs, axis=1)
```

We will also calculate Chroma Frequencies because they are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave. As in the other features, we will use librosa library to obtain them:

```
x, sr = librosa.load('../simple_piano.wav')

hop_length = 512

chromagram = librosa.feature.chroma_stft(x, sr=sr, 
hop_length=hop_length)
```

We will also use the time domain signal in order to calculate the Time Domain Zero Crossing, it will provide a measure of the noisiness of the signal and the formula that we will use will be:

$$Zt = \frac{1}{2} * \sum_{n=1}^{N} |sign(x[n]) - sign(x[n-1])|$$

Where the sign function is 1 for the positive arguments and 0 for the negative arguments and x[n] is the time domain signal for frame t.

The code in python, using librosa library for the formula will be:

```
zero_crossings = librosa.zero_crossings(x[n0:n1], 
pad=False)
```

We will detect the beat of the song using the library librosa and using the function beat.beat_track, we will get in return a float number as the estimated global tempo of an audio and an array with the estimated beat event locations in the specific units, the default would be what the frame indicates.

- **Evaluation.**

In order to evaluate the feature sets analysed before, standard statistical pattern recognition classifiers will be trained using real world data collected from a variety of different sources.

For classification purposes, we will use some statistical pattern recognition (SPR) classifiers. The idea behind a SPR is to estimate the probability density function for the feature vectors of each class. In the project we will use a existing classification algorithm to classify the songs into different genres such as the one that is in the next github repository: https://gist.github.com/parulnith/7f8c174e6ac099e86f0495d3d9a4c01e#file-music_genre_classification-ipynb

- **Light with genres.**

Once we have detected the genre of the song, we can proceed to send signals to the lights so they turn on or off depending on the genre that has been detected. We have decided to send the light module a matrix of bits to the light module determining the lights that have to be on or off and what color. The data will be sent by wifi, so we will have a sender in the control part a receptor in the light module, both connected to the same network.

Lights signals will variete depending on the detected genre. If we have detected a blues song, we will make the lights change slower than if we detected a rock song. If we detect a jazz song, the color of the lights will light with more warm colors, such as red, yellow or orange while an electronic song will light with more cold colors such as blue, white or purple.

2.11 Tolerance Analysis

A critical and challenging aspect to implementing the system is the detection module. Since the system's purpose is to work even in relatively noisy environments such as parties, examination of how well the detection module can perform is needed. In other words, can it record the music being played, filter out most of the low-frequency background noises, and send it to the control module with an acceptable accuracy?

In a living room with people talking, the noise level is usually at around 60dB. And the rock music being played at a house party can usually be around 70-75dB. The difference is 10-15dB. Take 10dB for example. This translates into a signal-to-noise power ratio of 10. In other words, the microphone will mostly be receiving the much louder rock music being played, instead of people's talking. Setting up the detection module closer to the music player can also increase the relative intensity of the music.

The fundamental frequency range of people talking is between 50-500Hz. The consonants can go up to 2k-4k Hz. The harmonics can even go beyond 12k Hz.[5] Generally, the needed range for rock music is about 60-8k Hz, with classical from 40-12k Hz.[4] Seems like there is a huge overlap between the two frequency spectra, but it is worth noting that the 250-2k Hz (low-mids) and 2k-8k Hz (high-mids) ranges are most important in the recognition of instruments, and thus of music. While it still has overlap with the fundamental frequencies of human voice between 250-500Hz, the major part of this frequency range is not affected. With a band-pass filter focusing more on the 250-8k Hz frequency range, attenuating the frequency ranges 0-250Hz and 8k-12k Hz, and filtering out the high-frequency part of the human speech noises above 12k Hz, the music contained in the audio signal can be more easily recognized.

3.COST AND SCHEDULE.

3.1 Cost Analysis.

3.1.1 Labor.

We estimate that our fixed development costs are $40/hour and 10 hours/week for 3 people, done over 16 weeks:

$$3 * \frac{\$40}{hour} * \frac{10\ hours}{week} * 16\ weeks * 2.5 = \$48,000$$

3.1.2 Parts.

Description	Manufacturer	Part #	Quantity	Cost
Raspberry Pi:a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse.	Premier Farnell		1	$35
USB Microphone for Raspberry Pi:1.5 m long cable; Omnidirectional response pattern; USB 2.0 (works with Raspberry Pi); 50 Hz - 16 kHz frequency response; Microphone Size (without windscreen): 6.5 cm x 0.7 cm; 44.1 kHz/48kHz USB Sample Rate Selection;	MakerPortal		1	$20
ASUS USB-N13 N300 USB 2.0 Wifi Adapter: Secure and Consistent connection - Powerful and designed for blazingly fast download, file transfer and media streaming. 2-in-1 device for wireless connection sharing - Enable	Asus		1	$11.95

your WLAN adapter into a wireless AP with design-in software AP PSP XLink Kai Support - Connecting PSP gamers all over the world EZ WPS - Wi-Fi configuration setting in just 2 steps with WPS hardware push button				
WiFi Module - ESP8266: 802.11 b/g/n Wi-Fi Direct (P2P), soft-AP Integrated TCP/IP protocol stack Integrated TR switch, balun, LNA, power amplifier and matching network Integrated PLLs, regulators, DCXO and power management units +19.5dBm output power in 802.11b mode Power down leakage current of <10uA 1MB Flash Memory Integrated low power 32-bit CPU could be used as application processor SDIO 1.1 / 2.0, SPI, UART STBC, 1×1 MIMO, 2×1 MIMO A-MPDU & A-MSDU aggregation & 0.4ms guard interval Wake up and transmit packets in < 2ms Standby power consumption of < 1.0mW (DTIM3)	SparkFun		1	$6.95

Microcontroller ATMEGA328P-PU			1	$2.08

3.1.3 Sum.

3.2 Schedule.

Week	Alfredo	Francis	Ran
3/2	Order parts for Detection and Wifi Module	Research genre detection	Order parts for the power module, light module
3/9	Test Amplifier, Microphone	Begin control module programming	Order more parts as needed, Submit Audit for PCB
3/16	Begin sound filter programming, test ADC	Continue control module programming	Test Power Module rectifier, batteries, charger
3/23	Continue sound filter	Finish control module programming, begin setting up PC UI	Solder Components onto PCB
3/30	Bugfix and finish sound filter, begin testing wifi Antenna	Finalize and bugfix control module programming, continue UI	Test wifi module microcontroller to interact with LED
4/6	configure wifi IC	Finish UI, Test control module with detection module inputs	Test Voltage regulator with all modules, test Wifi IC interface with control module
4/13	Test and finalize wifi modules	Test UI interface with control module	Finalize and bugfix UI interface with Control Module
4/20	Mock Demo/Finalizing	Mock Demo/Finalizing	Mock Demo/Finalizing
4/27	Demo	Demo	Demo
5/4	Prepare final presentation	Prepare final presentation	Prepare final report

3. ETHICS AND SAFETY

One concern is our usage of lithium ion batteries. When cell damage occurs within these batteries, a chemical fire can occur [9]. In order to combat this, we will be taking additional fire safety training and create a circuit within our power module to prevent the battery from either decaying below 3.0 V/cell or exceeding 4.2 V/cell. We will ensure that the battery will not be overcharged or over discharged, and prevent excessive heat from reaching the battery. Another issue is in charging the battery. Due to the nature of our project, a fully featured charging suite can be used to ensure that the circuit remains stable at all times. If costs do become a larger concern, we will use an integrated circuit solution and ensure no shorts or instabilities occur.

The general goal of both the IEEE Code of Ethics [7] and the ACM Code of Ethics [8] is to ensure quality without either intentionally or unintentionally causing harm. Our design does not appear to break any laws; the device will record audio only for the purposes of creating lights, and will not invade anyone’s privacy. The only wireless connections are made via bluetooth. This ensures little to no possibility of invading personal privacy or interfering with other signals, as our data and information is never collected into the internet.

Every subsystem aside from the lights and the microphone are contained within one compartment, which receives controlled inputs that would not cause overheating or other damage to the subsystems. The lights and microphone would be similar to ones you have at a normal home, so the ethical implications of those would be comparable as well. Finally, the lights are LED so they don’t require much power and create little risk.

We have attended basic safety lab training in order to learn how to use equipment while avoiding electrical shorts, shocks, and burns.

Overall, we believe that we are following both Codes of Ethics, as we are not breaching any regulations or standards. We will keep careful note to prevent our primary controller from overheating, but otherwise no safety concerns or breaches of privacy arise.

REFERENCES

[1] “Teach Tough Concepts: Frequency Domain in Measurements”, National Instruments, Aug-2018. [Online]. Available: http://www.ni.com/tutorial/13042/en/. [Accessed: 22-Feb-2020].

[2] “Musical genre classification of audio signals”, IEEE Transactions on speech and audio processing, 17-Nov-2002. [Online]. Available: https://dspace.library.uvic.ca/bitstream/handle/1828/1344/tsap02gtzan.pdf?sequence=1. [Accessed: 24-Feb-2020]

[3] “LibROSA”, librosa development team,© 2013-2019. [Online]. Available: https://librosa.github.io/librosa/.[Accessed: 23-Feb-2020].

[4] “Different Music, Different Speakers?”, Ohmspeaker.com , 16-Sept-2014. [Online]. Available: https://ohmspeaker.com/news/different-music-different-speakers/.[Accessed: 26-Feb-2020].

[5] “Facts about Speech Intelligibility”, dpamicrophones.com , 20-Jan-2016. [Online]. Available: https://www.dpamicrophones.com/mic-university/facts-about-speech-intelligibility.[Accessed: 26-Feb-2020].

[6] “Frequency Chart - The Most Important Audio Frequency Charts”, musicproductiontips.net [Online]. Available: https://musicproductiontips.net/wp-content/uploads/pdf/musicproductiontips.net-Frequency_Chart__The_Most_Important_Audio_Frequency_Ranges-letter.pdf .[Accessed: 26-Feb-2020].

[7] “IEEE Code of Ethics,” *IEEE*, Jun-2019. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 12-Feb-2020].

[8] “Code of Ethics,” *Code of Ethics*, 22-Jun-2018. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 12-Feb-2020]

[9]“Safe Practice for Lead Acid and Lithium Batteries,” University of Illinois ECE445 CourseStaff, 2016. [Online]. Available: htps://courses.engr.illinois.edu/ece445/documents/GeneralBatterySafety.pdf. [Accessed: 26-Jan-2020].