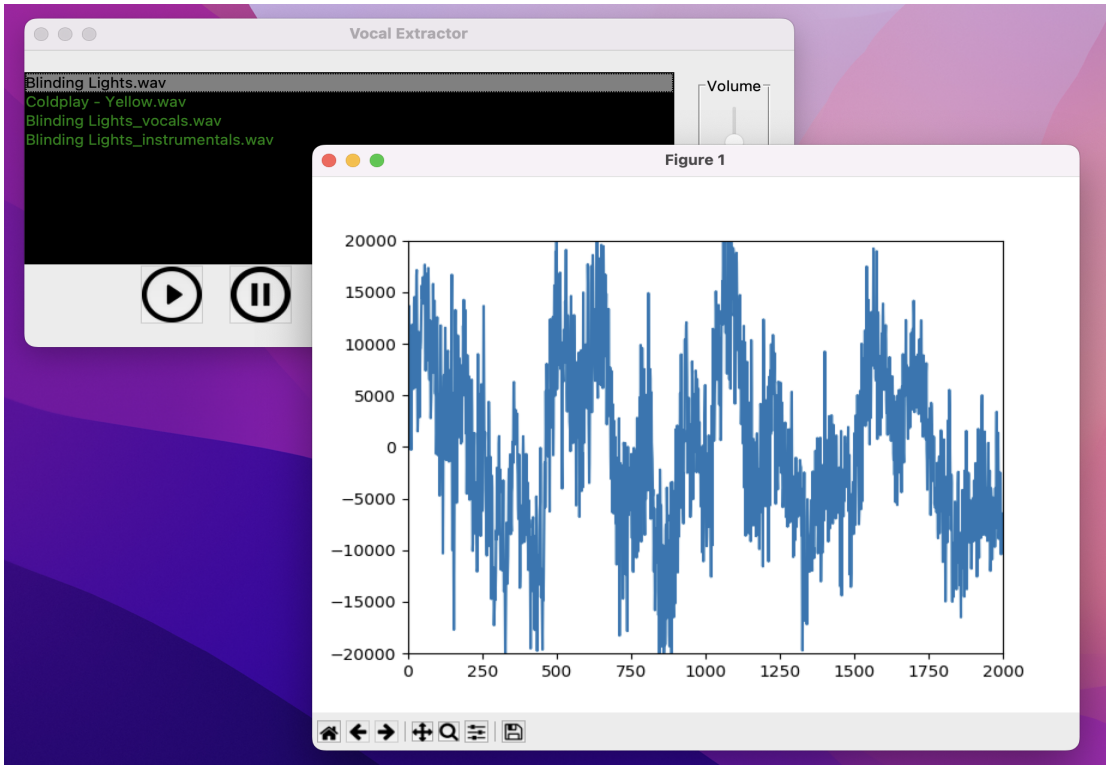
Vocals Extractor: A DSP Project

Extract vocals and instrumentals of a song at the click of a button

View Live Demo » | **View Source Code »**

Siddhanth Iyer (si2152) · Shikhar Vaish (sv2270)



About The Project

This is a submission to the course Digital Signal Processing Laboratory ECE-GY 6183 as the final project. The goal was to use signal processing techniques learned in the course and build something real-life oriented out of it. The application presented here executes a predefined audio processing algorithm to extract vocal and instrumental audio signals from a song. It assumes that the input audio file provided is musical audio with human voices.

(back to top)

Getting Started

Prerequisites

Dependency	Link
Python	Official Page

Installation

1. Download and extract the project zip.
2. Run the following command to install project dependencies: `pip install -r requirements.txt`

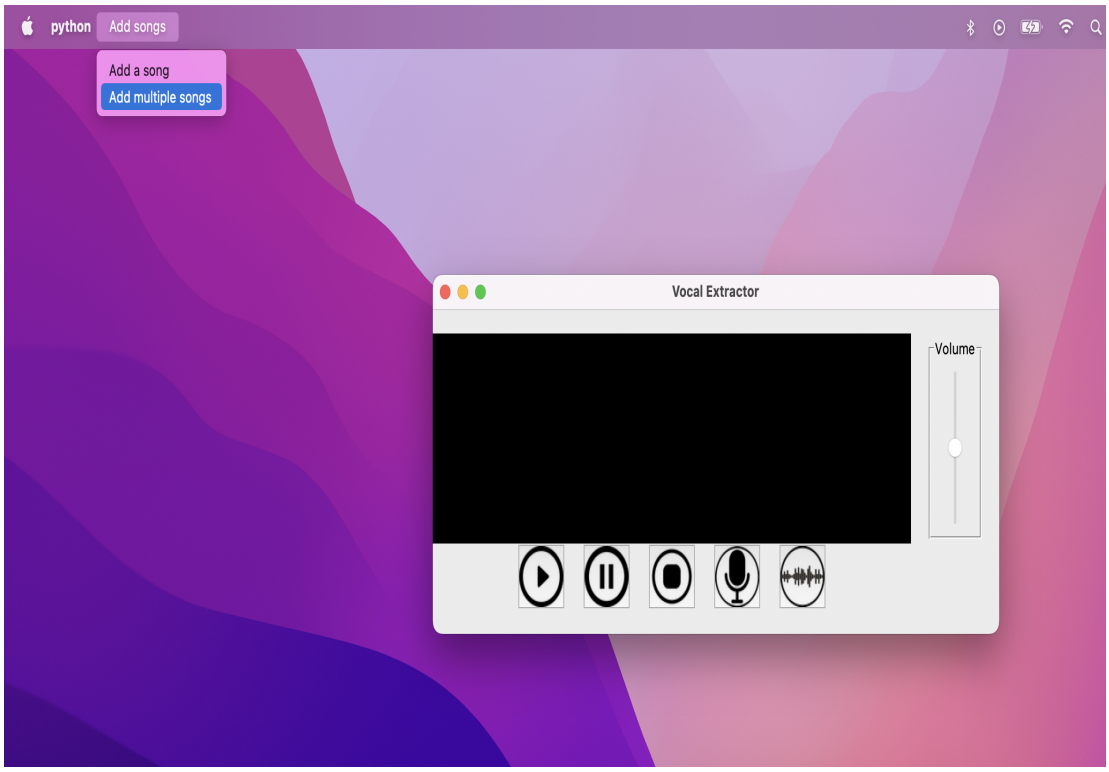
Run Application

1. Start the application using: `python main.py`

(back to top)

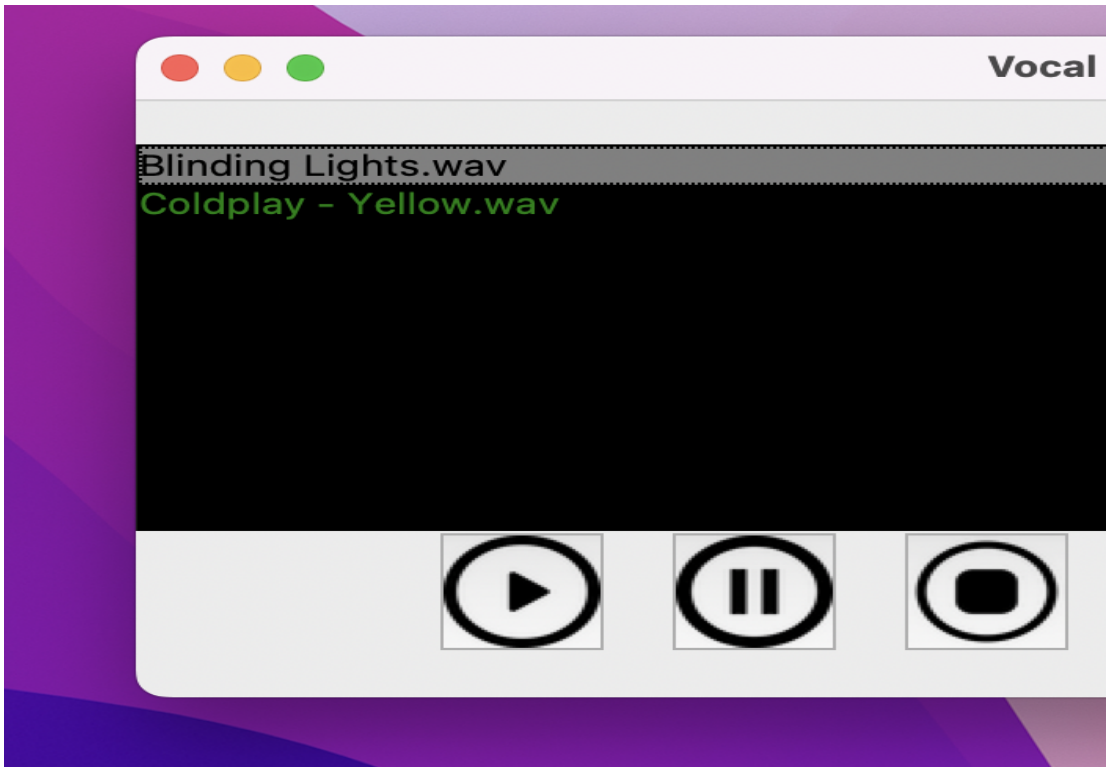
Usage

Load Songs



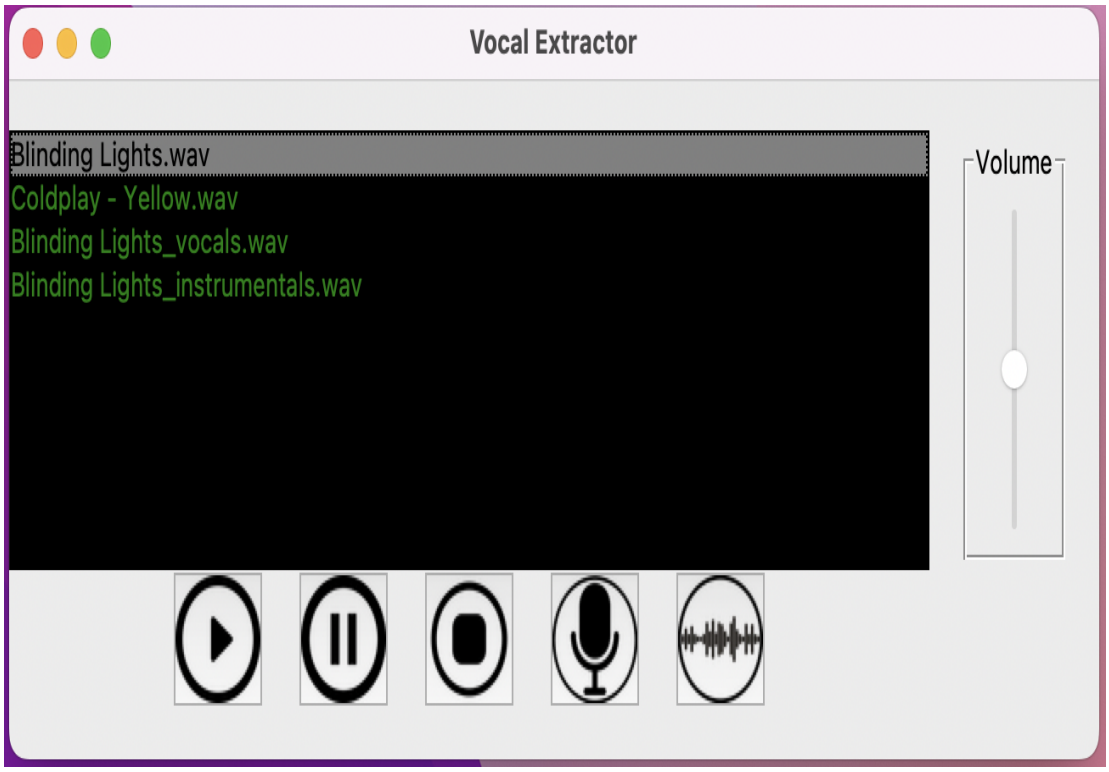
In the application menu, go to `Add Songs > Add a song` to open the file chooser pop-up. Select the file present on your computer to load it into the application. Intuitively, you can choose `Add Songs > Add multiple songs` to load multiple audio files.

Audio Control



You can use the intuitive control buttons to play, pause, and, stop playing the songs. The slider on the right can be used to adjust the playback volume.

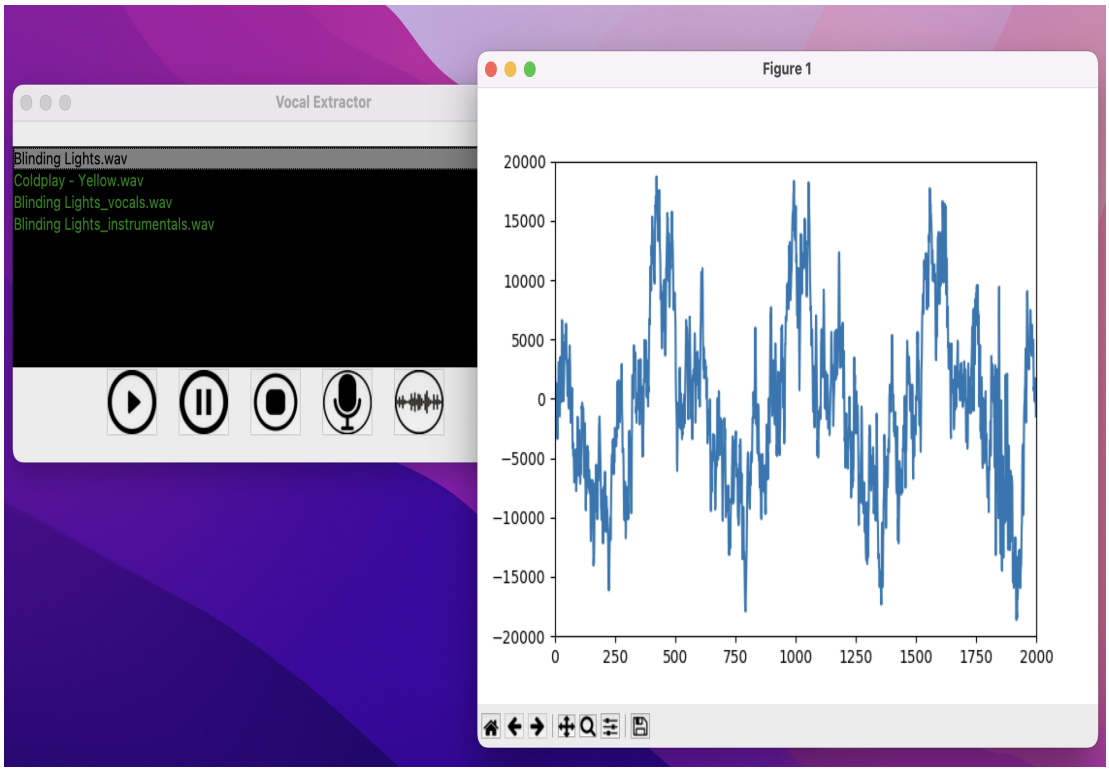
Extract Vocals and Instrumentals



First, click on the audio track that you want to process. Next, click on the microphone icon to extract the vocals and instrumentals of the selected audio file in separate files. It takes a few seconds to process the audio file, depending on the length of the audio track. Finally, you will see 2 new audio files saved as `<audio file name>_instrumentals.wav` and `<audio file name>_vocals.wav` in the `songs/` directory.

These audio tracks would be loaded into the application directly after processing. You can see listen to them directly through the application.

Real-Time Audio Visualization



Click the wave icon to visualize audio frequency in real-time.

How It Works

Audio Extraction

The application uses `librosa` library and begin with filtering audio signal by using nearest-neighbors filter. Each data point is replaced by aggregating its nearest neighbors in feature space. This can be useful for de-noising a spectrogram or feature matrix.

```
S_filter = librosa.decompose.nn_filter(S_full,
                                       aggregate=np.median,
                                       metric='cosine',
                                       width=int(librosa.time_to_frames(2,
sr=sr)))
```

Next, we apply 2 different softmask filters on the audio data, for instrumental and vocals respectively.

```
mask_i = librosa.util.softmask(S_filter,
                                       margin_i * (S_full - S_filter),
                                       power=power)

        mask_v = librosa.util.softmask(S_full - S_filter,
```

```
                                    margin_v * S_filter,
                                    power=power)
```

Finally, we generate the processed audio signals when we apply Inverse short-time Fourier transform(ISTFT) on the filtered signals.

```
y_foreground = librosa.istft(S_foreground * phase)

y_background = librosa.istft(S_background * phase)
```

Here, `y_foreground` contains the vocals and `y_background` contains the instrumentals. These signals are saved to new `.wav` files in the `/songs` directory with the suffixes `_vocals` and `_instrumentals`, respectively.

Real-Time Audio Visualization

The application uses `matplotlib` and a simple sliding window algorithm to load the binary data in blocks of fix sizes.

Here, the `BLOCKSIZE` is set as `2000`. We can change this, as per the application requirement and use-case.

Finally, we process the audio signal in batches and visualize, while playing the track, in real-time:

```
# Get block of samples from wave file
        input_bytes = wf.readframes(BLOCKLEN)

        while len(input_bytes) >= BLOCKLEN * WIDTH:

            # Convert binary data to number sequence (tuple)
            signal_block = struct.unpack('h' * BLOCKLEN, input_bytes)

            g1.set_ydata(signal_block)
            pyplot.pause(0.001)

            # Write binary data to audio output stream
            stream.write(input_bytes, BLOCKLEN)

            # Get block of samples from wave file
            input_bytes = wf.readframes(BLOCKLEN)
```

These batches are passed to `pyaudio` stream connected that plays the audio to the output device meanwhile we get matplotlib to read the audio data and plot the signals in real-time.

Authors

- Siddhanth Iyer - si2152
- Shikhar Vaish - sv2270